THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Visualize online collocation dictionary with force-directed graph

Bob Pham

This thesis is submitted in partial fulfillment of the requirements for the Degree of Master of Science at the University of Waikato.

Acknowledgements

I would like to thank my supervisor, Mark Apperley for his patient guidance and Shaoqun Wu for letting me use her collocational corpus.

TABLE OF CHAPTERS

Table Of Figures

TABLE OF TABLES:

Abstract

For second-language learners, collocational knowledge is very important. Knowing collocational phrases allows learners to speak and write in their targeted language naturally and reduce dramatically side effect of their first language. In order for learners to learn collocations easily, a lot of learning methods have been introduced. Particularly, learning from online-collocational corpus has become popular due to its accessibility and massive database. Although, its current presentation of information is still simple, it can be improved by using optimized representations in order to help users learning.

In this thesis, we represent a suitable way to visualize online collocational dictionary by using graph representation in order to facilitate users' learning and provide flexible exploration. Animation is also used to increase level of engagement for users. We use force-directed model for the layout, but we develop our own graph component and combine some current algorithms in order to create a proper algorithm for our purposes. The implementation is tested by a small group of participants and the results are promising.

Chapter 1 –Introduction

1.1.Chapter overview

In the first chapter, we introduce how important visualization information is and how human beings tend to look for potential information. Interactive methods and animation are also mentioned. Their effects on information visualization and users' mind are explained in detail. We also introduce graph visualization and its application on other areas.

Moreover, in this chapter, collocational knowledge is represented in detail. We also explain how collocational knowledge is important to second language learners. Major disadvantages which these users have to tackle are also explained in detail, especially "side effect of the mother language" consciously and unconsciously on their learning process. Online collocational corpus is introduced completely by displaying a typical example of an online collocational dictionary.

In second chapter, we look at literature. We introduce force-directed model, the role of force-directed model in graph visualization. Some current popular force-directed algorithms, such as Eades's or FR algorithms, are addressed in detail in order to show their advantages and drawbacks. They lead to our motivation to develop our force-directed algorithm.

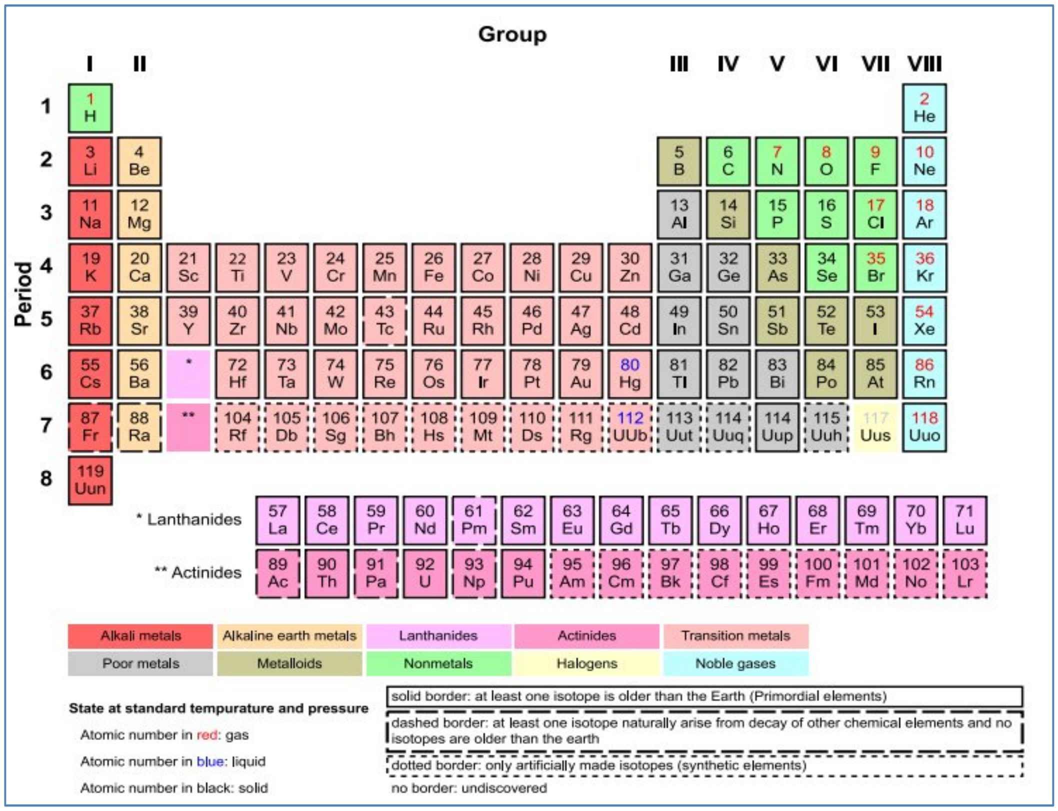
In third chapter, we discuss important requirements on graph layout from our database's structure and the way that users may interact with the system. In fact, these requirements prevent us from applying current force-directed algorithms. As a result of this, we develop our force-directed algorithm and edge component and explain them in detail in this chapter. Moreover, we also show fully our interface and all interactive techniques that we implement. Performance tests and usability tests are mentioned.

In the fourth chapter, we represent our conclusion and future work. And fifth is our reference list.

1.2.Introduction to Information Visualization

1.2.1. Information Visualization

People usually say “a picture is worth of a thousand words”, it means that complicated information can be conveyed with just a single image. Long time ago, people already knew how clear and effective when representing data on a drawing. Let consider a classic information application which is Mendeleev’s periodic table of elements (Figure 1.1 below).



Group
I II III IV V VI VII VIII
Period 1 2 3 4 5 6 7 8
1 H
2 He
3 Li 4 Be 5 B 6 C 7 N 8 O 9 F 10 Ne
11 Na 12 Mg 13 Al 14 Si 15 P 16 S 17 Cl 18 Ar
19 K 20 Ca 21 Sc 22 Ti 23 V 24 Cr 25 Mn 26 Fe 27 Co 28 Ni 29 Cu 30 Zn 31 Ga 32 Ge 33 As 34 Se 35 Br 36 Kr
37 Rb 38 Sr 39 Y 40 Zr 41 Nb 42 Mo 43 Tc 44 Ru 45 Rh 46 Pd 47 Ag 48 Cd 49 In 50 Sn 51 Sb 52 Te 53 I 54 Xe
55 Cs 56 Ba * 72 Hf 73 Ta 74 W 75 Re 76 Os 77 Ir 78 Pt 79 Au 80 Hg 81 Tl 82 Pb 83 Bi 84 Po 85 At 86 Rn
87 Fr 88 Ra ** 104 Rf 105 Db 106 Sg 107 Bh 108 Hs 109 Mt 110 Ds 111 Rg 112 UUb 113 Uut 114 Uuq 114 Uup 115 Uuh 117 Uus 118 Uuo
119 Uun
* Lanthanides 57 La 58 Ce 59 Pr 60 Nd 61 Pm 62 Sm 63 Eu 64 Gd 65 Tb 66 Dy 67 Ho 68 Er 69 Tm 70 Yb 71 Lu
** Actinides 89 Ac 90 Th 91 Pa 92 U 93 Np 94 Pu 95 Am 96 Cm 97 Bk 98 Cf 99 Es 100 Fm 101 Md 102 No 103 Lr
Alkali metals
Alkaline earth metals
Lanthanides
Actinides
Transition metals
Poor metals
Metalloids
Nonmetals
Halogens
Noble gases
State at standard tempurature and pressure
Atomic number in red: gas
Atomic number in blue: liquid
Atomic number in black: solid
solid border: at least one isotope is older than the Earth (Primordial elements)
dashed border: at least one isotope naturally arise from decay of other chemical elements and no isotopes are older than the earth
dotted border: only artificially made isotopes (synthetic elements)
no border: undiscovered

Figure 1.1: A basic simple of Mendeleev’s periodic table of the elements.
(Image taken from Steele & Iliinsky (2010))

By studying chemistry in high school, we understand that elements in the table have properties that repeat periodically. In detail, Mendeleev arranged them into proper rows and columns to order to indicate the periodicity of these properties. Importantly, the arrangement also reveals the related, repeating physical properties of the elements. Specially, he also showed his very accurate predictions of undiscovered elements by reserving empty places on the table.

The periodic table of the elements is indeed informative and efficient. This neat organization not only allows viewers to access quickly a given element and its periodic properties, but also indicates relationships between these relational complex elements which we might need thousands of words if it were explained by words.

Another notable example which shows the powerful of drawings is “The London underground map” designed by Harry Beck in 1933 (Steele & Iliinsky, 2010) (Figure 1.2 below). In fact, the author mainly used simple graphic components such lines, colors and shapes, but the drawing is clear and full of information and seems impossible to be explained just by words. Steele & Iliinsky (2010) mention that “the map highlighted the most relevant information and stripped away much of the irrelevant information, making the pertinent data more easily accessible”. This novel example is praised to have a distinctive and unique graphical style and widely recognized as a masterpiece of information visualization.

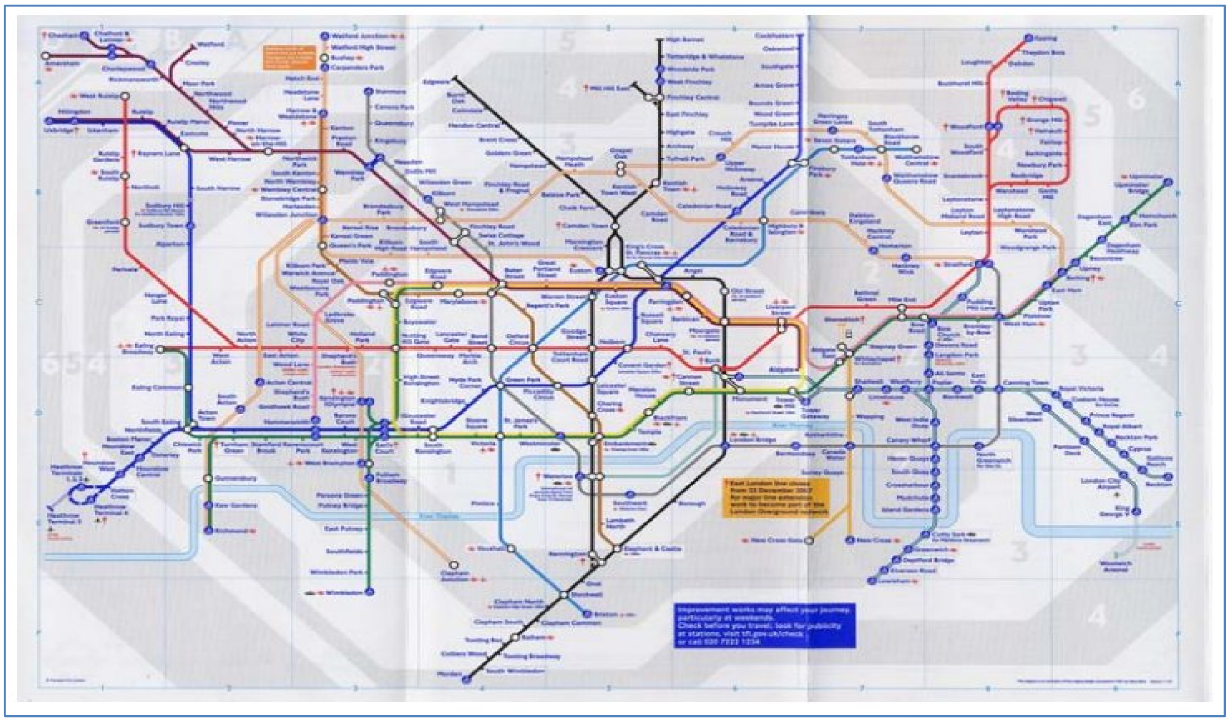


Figure 1.2: The London underground map (2007 version).
(Image taken from Steele & Iliinsky (2010))

The two classical examples above are typical evidence to prove that how important practical information visualization is in our lives. In fact, information visualization has been a new distinctive area of research. Because of its great effect on other areas such as digital libraries, data mining and human-computer interaction, it quickly becomes an interdisciplinary research area (Chen, 2004). Unlike other topics such as data mining which is involved a lot in pattern-finding algorithms and computer-based-computation, or programing languages which is involved strongly in computer science theory, information visualization, on the other hand, is involved in studying and making use of human perception and concerns about viewers' cognition (Ware, 2004).

In fact, information visualization quite relates to human-computer interaction. User-interface designer always concern about information displayed. Chen (2004) also pointed out that "Information visualization represents one of the latest streams in a long established trend in modern user interface. The desire to manipulate objects on a computer screen has been the driving force behind many popular user interface design paradigms". Notable examples are the "what-you-see-is-what-you-get" and "drag-and-drop" user interfaces (Chen, 2004).

Information visualization has quite a number of definitions. Stuart, Jock and Shneiderman (1999) defined that "Information visualization is the use of computer-supported interactive visual representations of abstract data to amplify cognition". Cognition is the acquisition of knowledge. In case of a chart, the cognitive purposes here are to know which records have max or min values and importantly the trend which reflects "the whole picture" of data. In case of a map, the cognitive problem here is to locate a specific location and how to get from the current one to the destination (Figure 1.3 below).

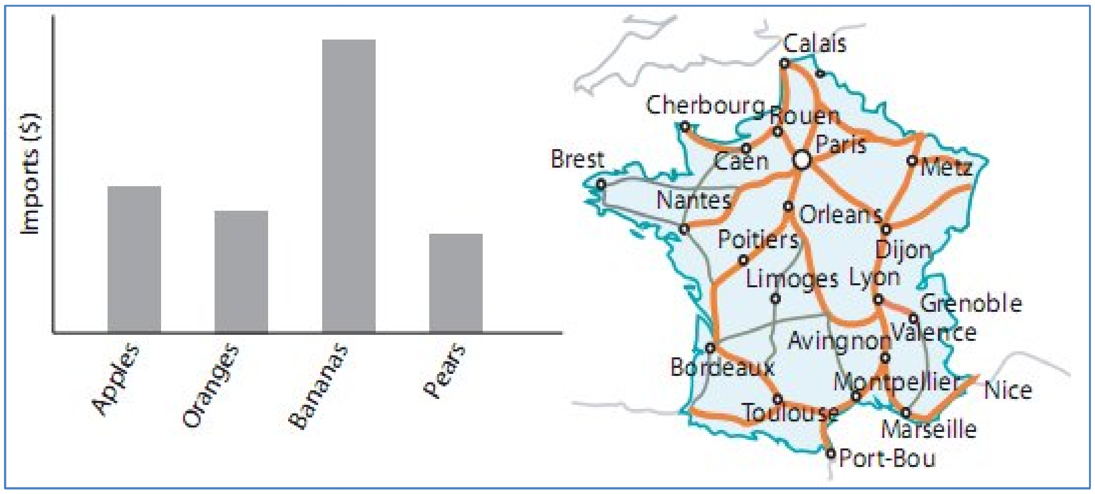


Figure 1.3: Illustrate cognitive problems in a chart and map applications
(Image taken from Ware (2008))

However, in order to make an effective visual representation, designers have to study and make use of human perceptions. In addition, Ware (2008) provides us a better and detailed explanation how information visualization augments human cognition. He explains that humans use their eyes to sample the visual world as a very natural basic activity. For cognitive purposes, humans pay attention to pieces of potential information. Then, their brains will extract patterns which are useful and relevant to their purposes and the patterns will be hold briefly in their working memory. He defines important terms "visual thinking" and "visual queries". "Visual thinking consists of a series of acts of attention, driving eye movements and tuning our pattern-finding circuits. And these acts of attention are called visual queries" (Ware, 2008). With that understanding, information visualization role is to exploit human perception to distinguish potential useful information, making "visual queries" easier and alleviating pattern-finding process by acting as an external memory.

1.2.2. Interacting with Visualizations

a) Interactive techniques

From the definition of Information visualization of Stuart, Jock and Shneiderman (1999), they pointed out that applications of information visualization should support interaction. Moreover, according to Ware (2004), he also mentioned that "a good visualization is not just a static picture that we can walk through and inspect like a museum full of statues. A good visualization is something that allows us to drill down and find more data about anything that seems important". Simply speaking, good information visualization should provide interactive techniques that support explorations and navigations.

Nowadays, with the help of computer, a lot of interesting interactions are created. For example, traditional interactive techniques are: zooming in to get details, zooming out to get an overview and panning to move views. Scrolling is also a popular navigating technique. It normally consists of a scroll-bar to control a viewport. Scrolling is commonly implemented in document viewer applications such as word processors. Even though, it allows viewers to access thoroughly documents, it still has some limits. For example, its showing area is relative smaller than the size of the document, so it gives no off-screen information. In addition, if the size of the document is particularly large compared to the viewport, the scrolling will become small which may reduce usability; moreover, the travelling speed increases rapidly, causing troubles of getting properly some positions. In fact, these techniques complement each other, making data access easily which directly speeds up finding-pattern process (Herman, Melancon and Marshall, 2000).

Zooming in and zooming in are considered as the two separate views; as a result of this, it can cause context loss when users change from one to another (Herman, Melancon and Marshall, 2000). In fact, there has been a lot of research in interactive techniques in order to overcome this usability obstacle. Notable examples are magnifying glass techniques and fisheye views. Basically, they merge an overall view and a detailed view into one single view which allows users to access details of a given area and at the same time maintain the overall context (Cockburn, Karlson and Bederson, 2008). These examples are typical applications of the interactive technique called "Focus + Context". Fisheye techniques can have different implementations due to a variety of distortion transformations; for example, Polar and Cartesian transformations (Cockburn, Karlson and Bederson, 2008).

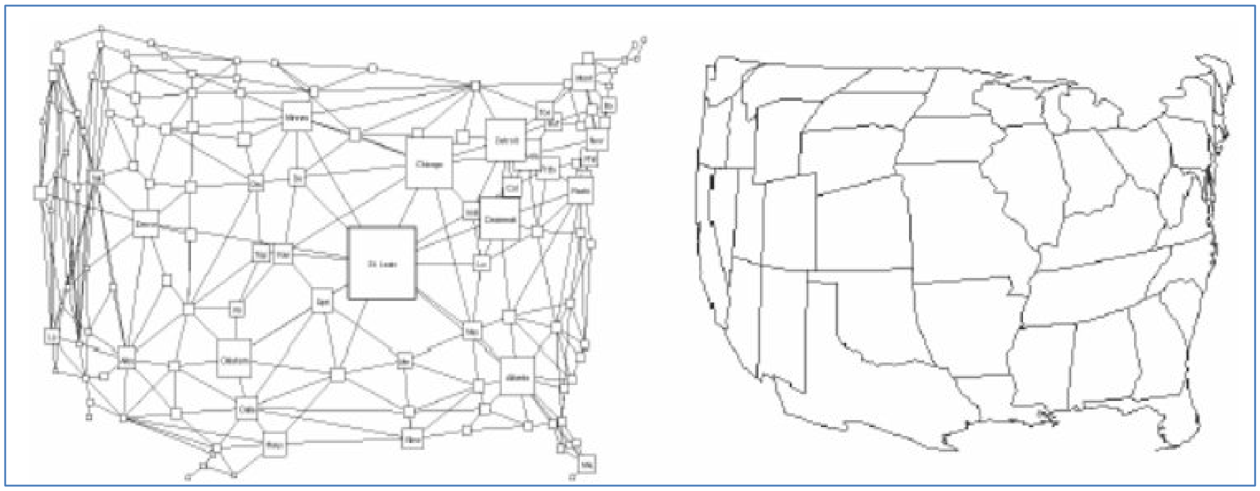


Figure 1.4: Left image: Fisheye with Cartesian transformation of a graph of major US cities.
Right image: Fisheye with a Polar transformation of a map of the US.
(Image taken from Cockburn, Karlson and Bederson (2008))

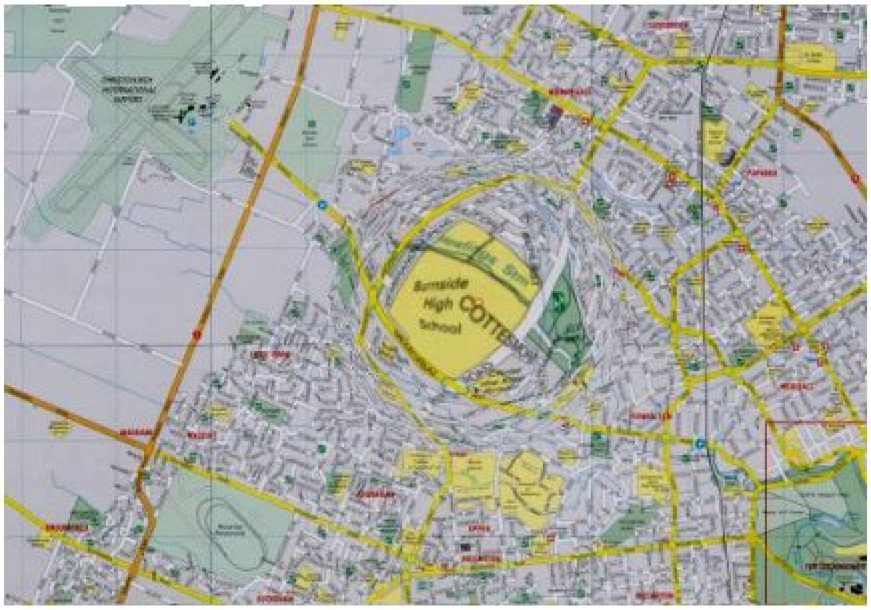
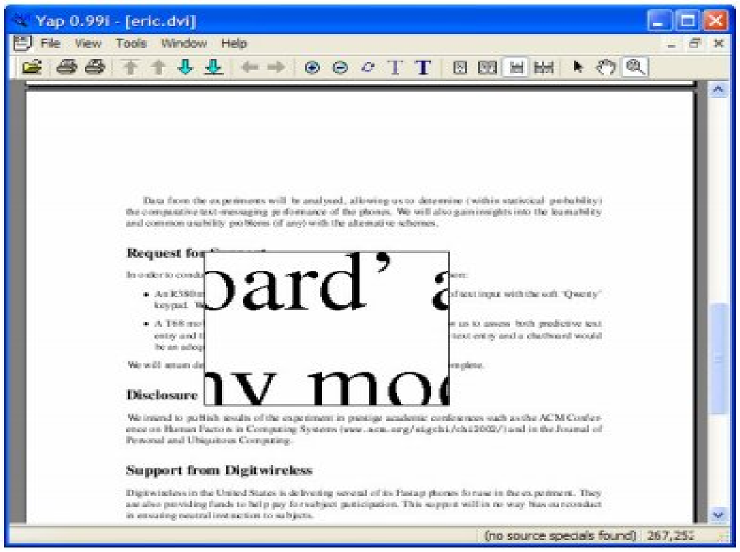


Figure 1.5: A fisheye view on a map.
The focal region is magnified and display with its surrounding context.
(Image taken from Cockburn, Karlson and Bederson (2008))


Yap 0.99l - [eric.dvi]
File View Tools Window Help
(no source specials found) 267,252

Data from the experiments will be analysed, allowing us to determine (within statistical probability) the comparative text-messaging performance of the phones. We will also gain insights into the learnability and common usability problems (if any) with the alternative schemes.

Request for

In order to cond [illegible]

- An R380 [illegible] of text input with the soft 'Qwerty' keypad. We [illegible]
- A T68 mo [illegible] us to assess both predictive text entry and t [illegible] text entry and a chatboard would be an adop [illegible]

We will attain de [illegible] complete.

Disclosure

oard' a

ny mod

We intend to publish results of the experiment in prestige academic conferences such as the ACM Conference on Human Factors in Computing Systems (www.acm.org/sigchi/chi2002/) and in the Journal of Personal and Ubiquitous Computing.

Support from Digitwireless

Digitwireless in the United States is delivering several of its Fastap phones for use in the experiment. They are also providing funds to help pay for subject participation. This support will in no way bias our conduct in ensuring neutral instruction to subjects.

Figure 1.6: Magnifying lens technique example.
(Image taken from Cockburn, Karlson and Bederson (2008))

b) Animation

With the rapid increase in power of computer hardware, animation has been used widely in interactive techniques. Intuitively, designers just aimed to use it in order to smooth transitions of data graphics, but according to several researches, it has more effect on the cognition of viewers.

In detail, according to Heer & Roberton (2007), animation not only connects the transitions between different parts of data graphics, but also keeps viewers oriented in order to prevent them from fast loosing focus on the current processed information. Particularly, Tversky, Morrison and Betrancourt (2002) also pointed out that “animation increases levels of engagement”, Gonzales (1996) mentioned that “animation improves decision making”, and Bederson & Boltman (1999) showed that “animation facilitate learning”.

In terms of visual queries and visual thinking (Ware, 2004), animation contains motion which is highly effective at attracting attention. As a result of this, it directly speeds up viewers’ visual queries. In addition, when animation runs, it is changing related data graphics with certain time for viewers to mentally prepare for incoming information and connect previous visual thinking with this information.

1.2.3. Graph Visualization

Graph is an important area of visualization information. A typical graph is represented as a node-link diagram; it contains nodes and edges, whereas nodes represent entities or data with edges representing relations or connections. According to Herman, Melancon and Marshall (2000), a simple way to determine that using graph visualization is suitable is to consider the following question: “is there an inherent relation among the data elements to be visualized?” If the answer is “yes”, then graph is the most suitable choice.

a) Design visualization on graph

In fact, for visualization purposes, the components of a graph such as nodes and edges can be used to denote different meanings. For example, in terms of nodes, using different colors on nodes to indicate different categorized subjects, domains or entities, and using different shapes on nodes to indicate different levels of importance (Figure 1.7 below).

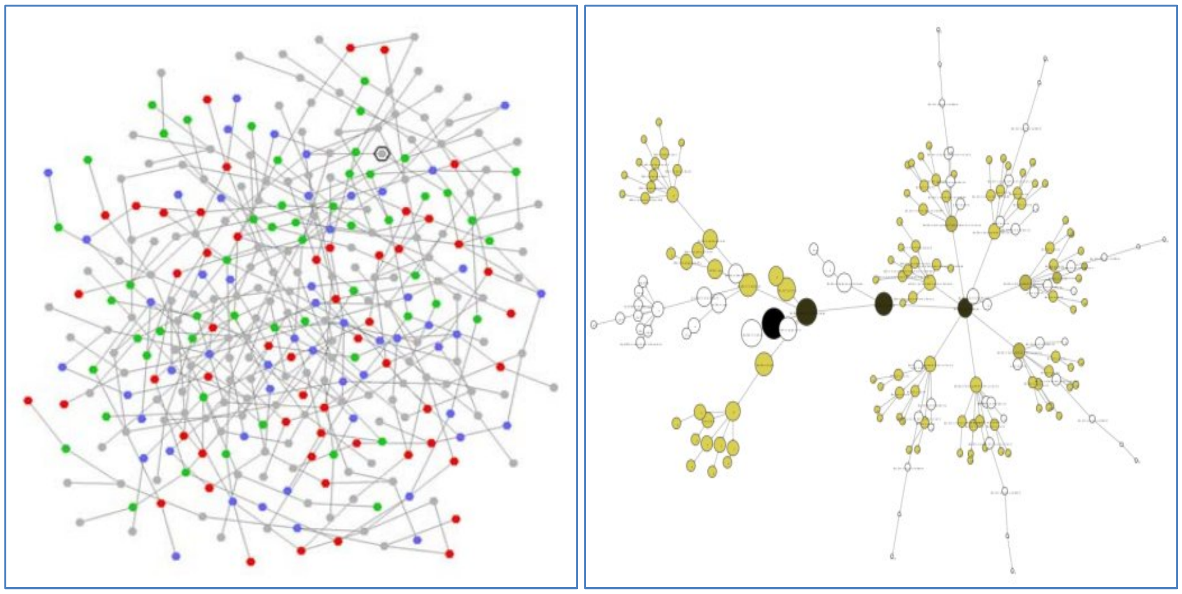


Figure 1.7: Graphs with encoded components
Left image: Graph with different color nodes. Right image: Graph with the size and color of nodes are encoded.
(Image taken from Herman, Melancon and Marshall (2000))

In terms of edges, they can be encoded to show direction or flow between two connected nodes, what kind of their relationship, or how strong it is. In detail, in order to display show or direction, a common design is to use an arrow edge; in order to display different relationships, colors are use and length or thickness can be used to show levels of the relationship (Figure 1.8).

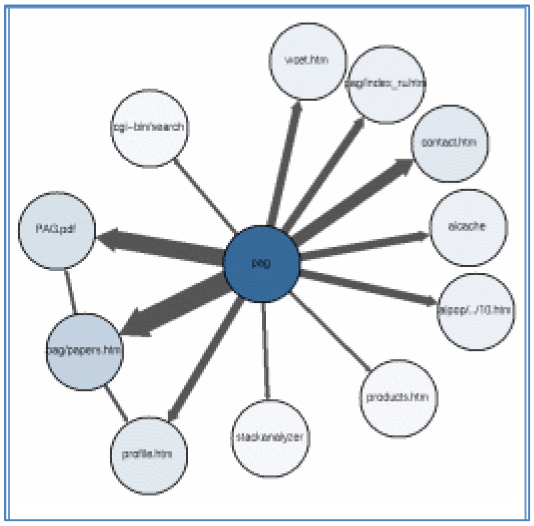


Figure 1.8: Graph with edges encoded.
Different encoded messages are used on one single edge.
(Image taken from Herman, Melancon and Marshall (2000))

b) Typical application areas

Graph visualization is very practical and has many areas of application. The most common one that we can mention is the file hierarchy on a computer. In other scientific areas such as biology and chemistry, graphs are applied to visualize evolutionary trees, phylogenetic trees, molecular maps, genetic maps, biochemical pathways and protein functions (Herman, Melancon and Marshall, 2000). In software management, UML diagrams, data flow diagrams and entity relationship diagrams are also typical applications of graph. In network visualization, graph is considered as the most suitable choice. Recent years have showed the rising popularity of online social networking services such Facebook, and Twitter; as a result of this, applications for research of these online communities have become necessary. By visualizing these networks with a node-link layout, users are able to not only observe the whole picture of the networks, but also explore their connectivity. In addition, the node-link layout allows designers to implement a local search to analyze individual entities and different highlight techniques. Noble example can be mentioned is Vizster- a visualization tool for online social network developed by Heer and Boyd (2007).

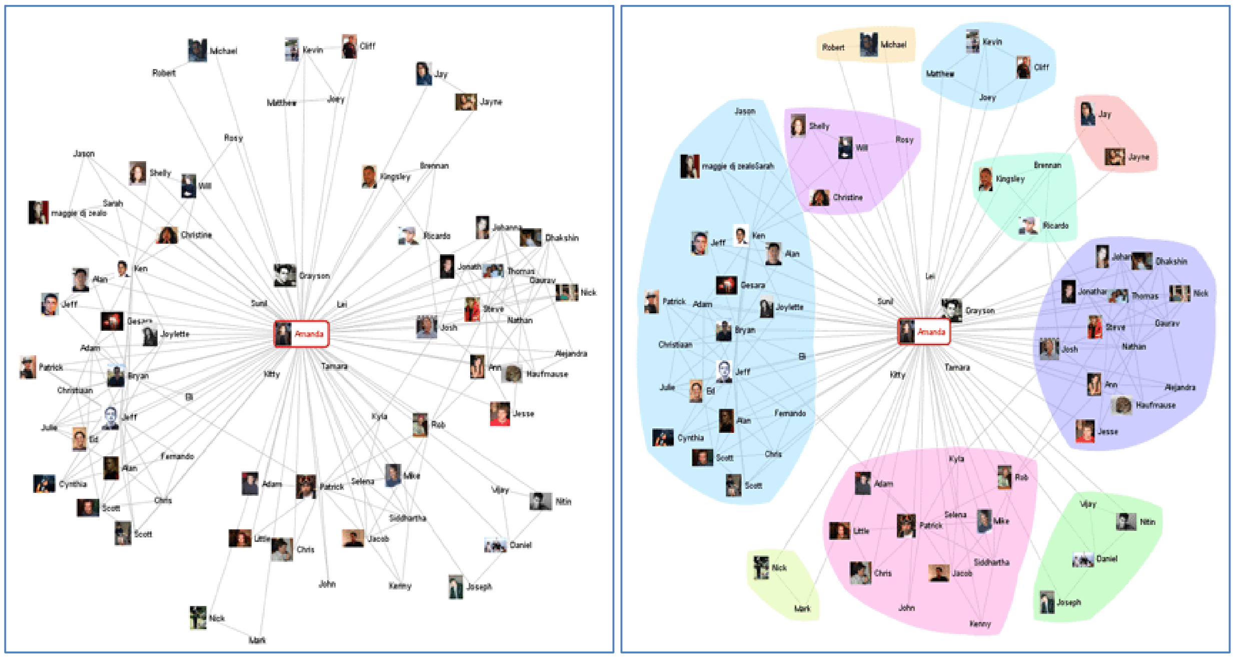
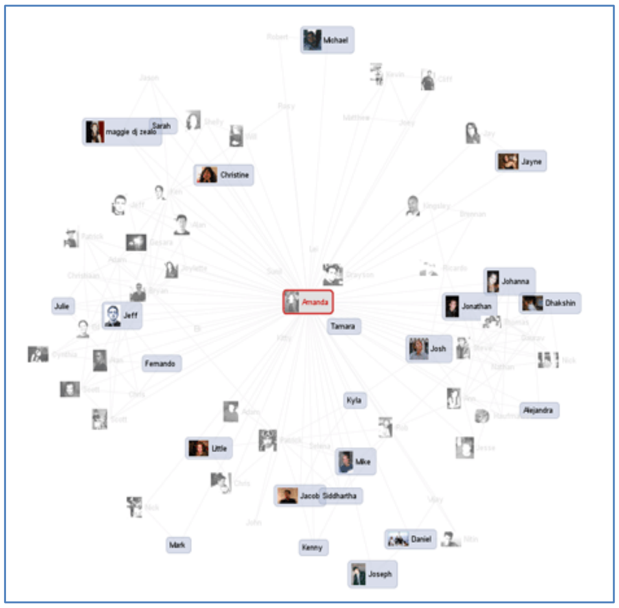


Figure 1.9: A social network visualized by Vizster.
Left image: Normal mode of the graph. Right image: Graph with highlighted groups
(Image taken from Heer and Boyd (2007))



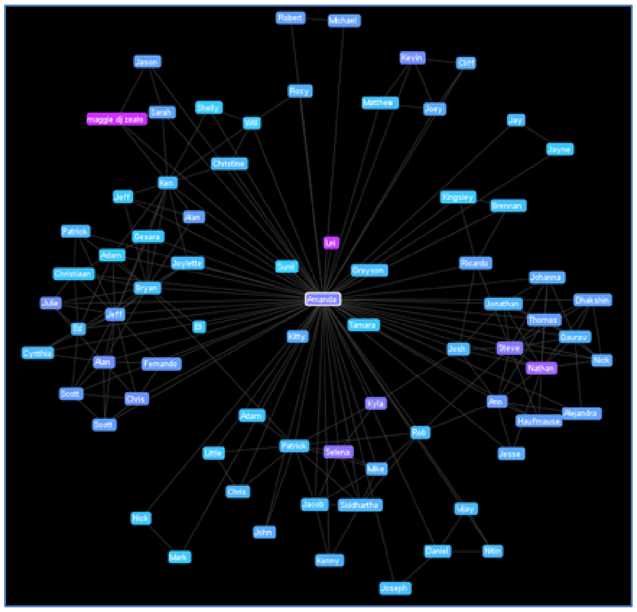


Figure 1.10: A social network visualized by Vizster with different modes
Left: Highlighted results of individual searches. Right: Graph with x-ray mode (links highlighted)
(Image taken from Heer and Boyd (2007))

1.3.Introduction to English Collocations

1.3.1. English Collocations

According to Lewis (1997), collocations in linguistics are defined as "arbitrarily restricted lexeme combinations that co-occur in natural text with greater than random frequency" (as cited in Nakata, 2006). Previous studies also categorize collocations into two groups which are: lexical collocations and grammatical collocations (Benson et al., 1986). Lexical collocation's structures are constructed from nouns, verbs, adjectives and adverbs. According to Benson et al (2006), there are six common structures which are:

Types of lexical forms	Examples
Verb + noun	Set a record, take care
Adjective + noun	Heavy rain, strong tea
Noun + verb	Cats jumps, plants grows
Noun + (of) + noun	A set of principles, a pack of cigarettes
Adverb + adjective	Arguably large, closely acquainted
Verb + adverb	Create properly, disagree strongly

Table 1.1: Table of lexical forms and their examples

On the other hand, grammatical collocations are phrases which contain a central word. It can be a verb, adjective or noun. In detail, if the central word is a verb or adjective, the flowing word will be a particular preposition; if the central word is a noun, then it must be followed by a particular form of the verb.

Grammatical collocations	Examples
Verb + Preposition	Take off, take on
Adjective + Preposition	Afraid of, interested in
Noun + Particular form of verb	Problems to solve, strength to lift it

Table 1.2: Table of grammatical forms and their examples.

Moreover, like other researchers, we only concern lexical aspect which is primary topic in linguistic research (Lewis, 1997, Liu, 1999 and Li, 2005).

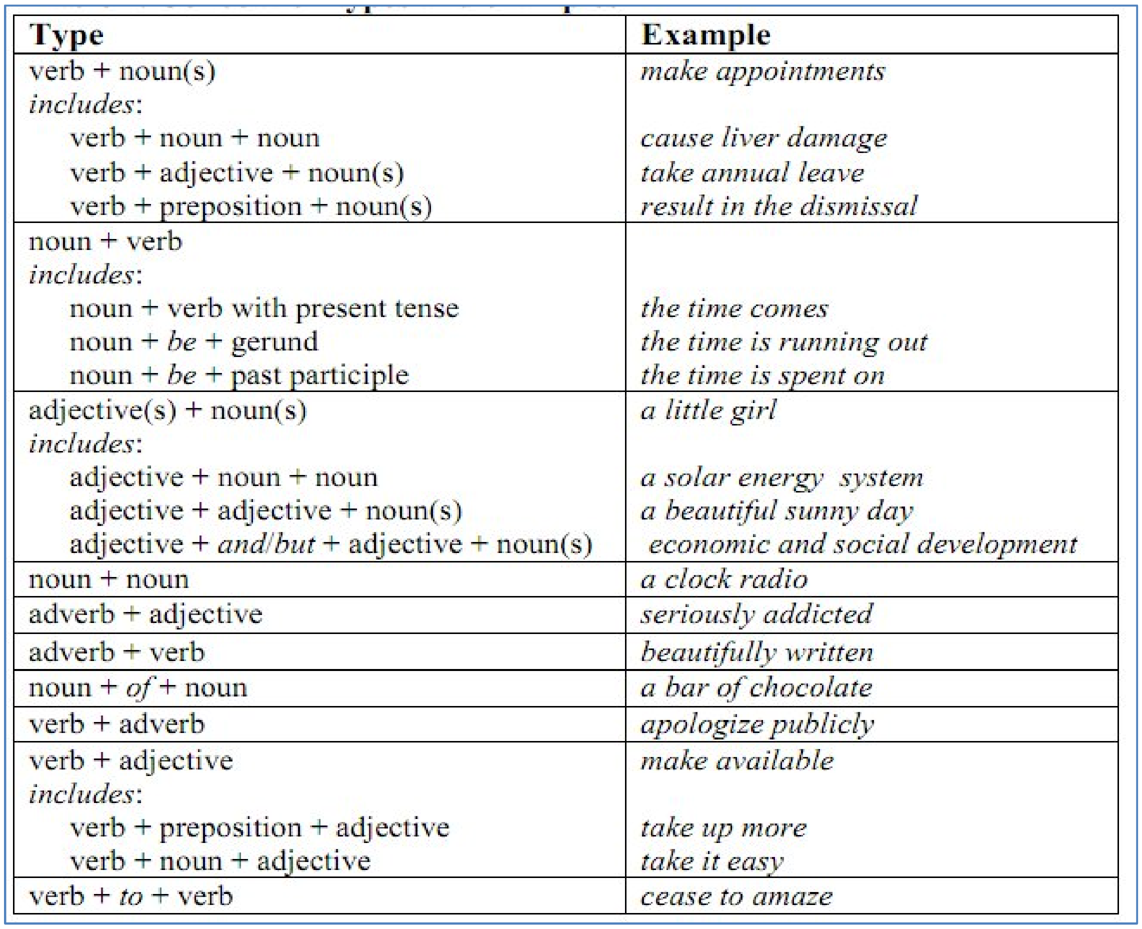
In fact, there have been a lot of studies conducted particularly in Asia such as Japan, Taiwan and China in order to measure collocational knowledge of English-second-language learners (L2) at high levels such as high school and university levels (Nakata, 2006; Li, 2005 and Wu, 1996). Most results showed that participants still made a lot of collocational errors in their writing and speaking for lack of proper collocational teaching in English classes. According to Wu (1996), students tended to learn English by looking up new words which they came across in textbooks, then memorized their meanings. This passive learning allows them comprehended text with help of related context; however, it hardly helped them to generate proper structures in speaking and writing (Wu, 1996).

Moreover, negative language transfer, which is judgment of learners on structures of the target language (L2) which are similar to their first language's, also has considerably effect on the second language acquisition (Chen, 2004 and Jiang, 2004). For example, Chinese learners rather use "take medicine" as "eat medicine" because the noun "medicine" in Chinese normally goes with the verb "eat" rather than "take" (Darvishi, 2011). On the other hand, Vietnamese students affected by their language, in some writing context falsely use "grow" instead of "build" for houses or buildings. In addition, in Japanese, they use "take contact" and "pay sacrifice" instead of using "make contact" and "make sacrifice" in English (Nakata, 2006). Previous studies showed that first language (L1) caused more collocational errors in some particular lexical structures; in case of Chinese language, "verb + noun" form was considered the lexical combination having highest using error rate (Wu, 1996 and Liu, 1999); in case of Japanese language, students were also prone to "verb + noun" structure (Nakata, 2006).

1.3.2. Web-derived corpus approach to collocations

Collocations have been received a lot of studies and are considered to be the primary mean for second language learners to produce language fluently. Nowadays, with the rising popularity of computer and internet, many new online methods of learning languages are developed and introduced. According to Peachey (2005), he defined "a concordance is a piece of software, either installed on a computer or accessed through a website, which can be used to search, access and analyze language from a corpus" (as cited in Wu, Witten, and Franken 2010). In fact, online dictionary has been very conveniently due to its easy accessibility.

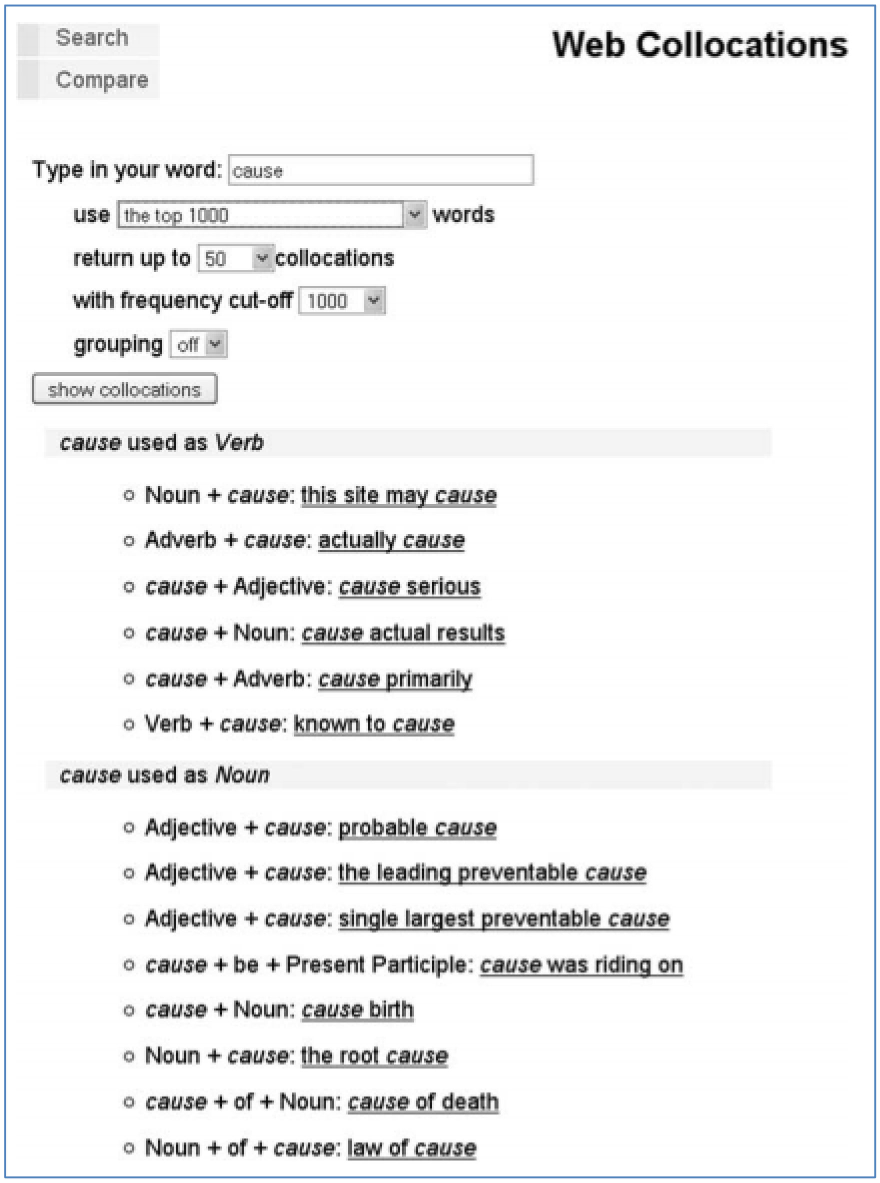
Moreover, in terms of studying collocations, web-derived corpus developed by Wu, Witten and Franken (2010) shows a lot of advantages. In detail, they used an off-line database—n-gram collection generated and supplied by Google. This database is considerably large, consists of different sized combinations from one word (1 gram) to 5 words (5-grams). As a result of this, it covers all useful lexical and grammatical forms. Particularly, frequency of every phrase is also recorded indicating how popular it is. In addition, this database is cleaned up and categorized into lexical structures. According to Wu, Witten and Franken (2010), they also used lexical patterns from the work of Benson, Benson and Ilson (1986) and lexical forms from Oxford Collocation Dictionary such as noun + noun, adverb + verb, verb + to + verb and verb + adjective. Moreover, they also selected some other common patterns. In total, the lexical forms are listed in Table 1.3.



Type	**Example**
verb + noun(s) *includes*: verb + noun + noun verb + adjective + noun(s) verb + preposition + noun(s)	*make appointments* *cause liver damage* *take annual leave* *result in the dismissal*
noun + verb *includes*: noun + verb with present tense noun + *be* + gerund noun + *be* + past participle	 *the time comes* *the time is running out* *the time is spent on*
adjective(s) + noun(s) *includes*: adjective + noun + noun adjective + adjective + noun(s) adjective + *and/but* + adjective + noun(s)	*a little girl* *a solar energy system* *a beautiful sunny day* *economic and social development*
noun + noun	*a clock radio*
adverb + adjective	*seriously addicted*
adverb + verb	*beautifully written*
noun + *of* + noun	*a bar of chocolate*
verb + adverb	*apologize publicly*
verb + adjective *includes*: verb + preposition + adjective verb + noun + adjective	*make available* *take up more* *take it easy*
verb + *to* + verb	*cease to amaze*

Table 1.3: Collocation types and examples.
(Table taken from Wu, Witten and Franken (2010))

Furthermore, in order to look up collocations, users just locate the search page and enter a word. Figure 1.11 below shows that when the user enters the word "cause" and it returns possible lexical forms for the two categories "used as Verb" and "used as Noun".



Type in your word: cause
use the top 1000 words
return up to 50 collocations
with frequency cut-off 1000
grouping off

cause used as *Verb*

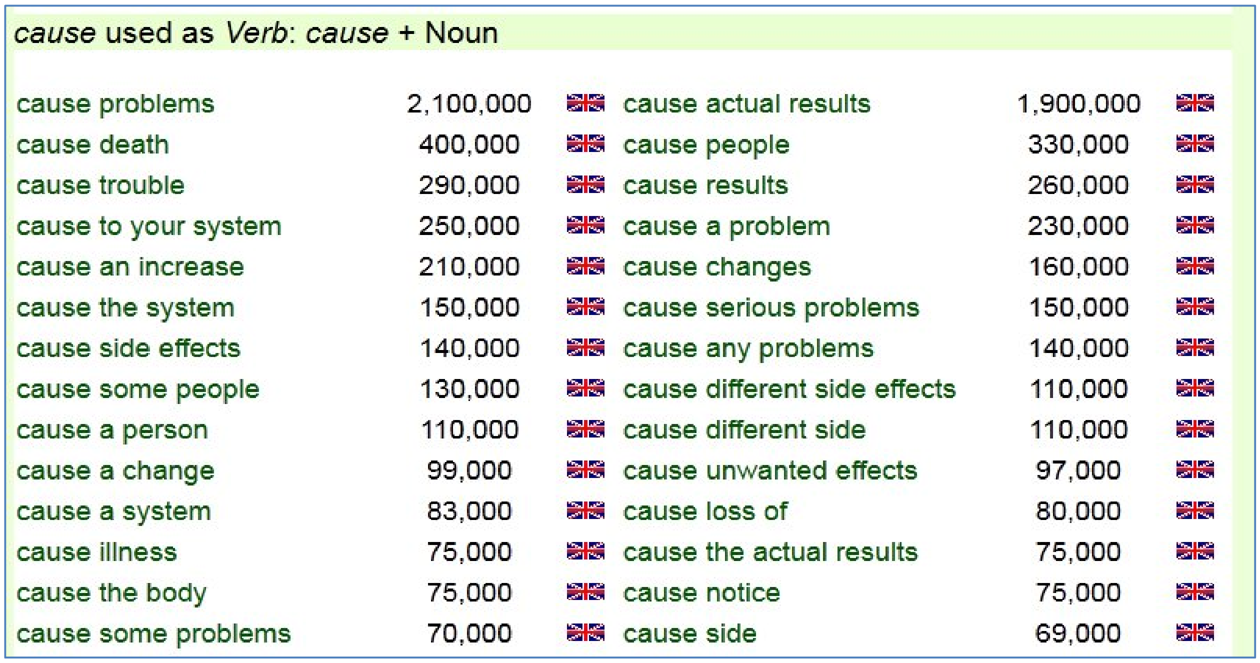
- Noun + *cause*: this site may *cause*
- Adverb + *cause*: actually *cause*
- *cause* + Adjective: *cause* serious
- *cause* + Noun: *cause* actual results
- *cause* + Adverb: *cause* primarily
- Verb + *cause*: known to *cause*

cause used as *Noun*

- Adjective + *cause*: probable *cause*
- Adjective + *cause*: the leading preventable *cause*
- Adjective + *cause*: single largest preventable *cause*
- *cause* + be + Present Participle: *cause* was riding on
- *cause* + Noun: *cause* birth
- Noun + *cause*: the root *cause*
- *cause* + of + Noun: *cause* of death
- Noun + of + *cause*: law of *cause*

Figure 1.11: An example of Wu's software interface.
(Image taken from Wu, Witten and Franken (2010))

Moreover, if the user clicks on one of these clauses such as "cause + Noun", it will link to another site showing results of phrases sorted by descending frequencies. Figure 1.12 bellows show top results of this lexical form from n-gram Google database.

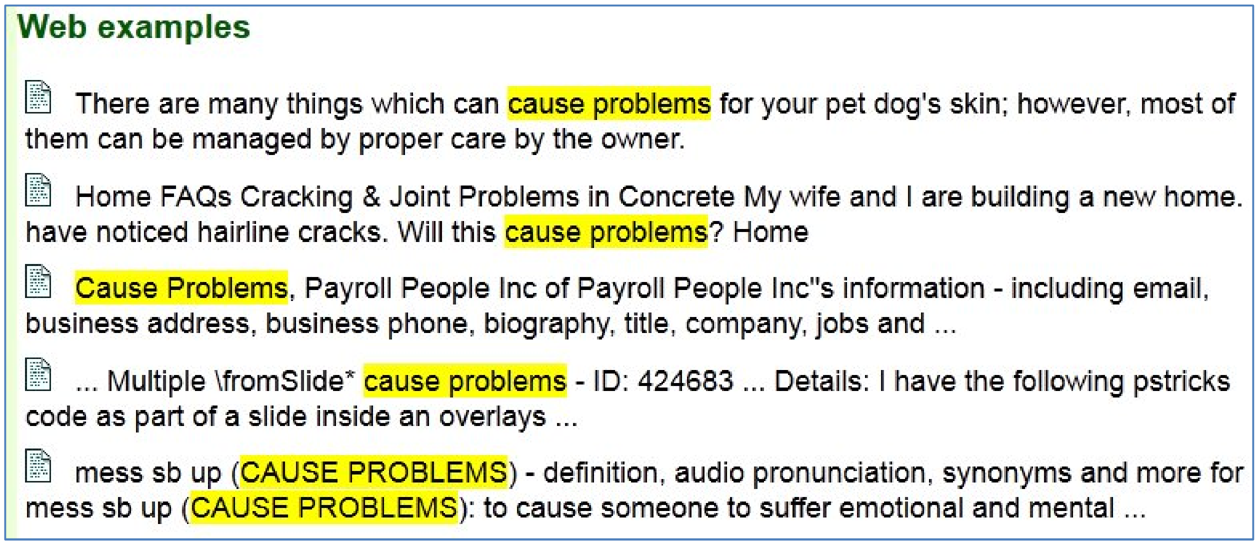


cause used as *Verb*: *cause* + Noun

cause problems	2,100,000	🇬🇧	cause actual results	1,900,000	🇬🇧
cause death	400,000	🇬🇧	cause people	330,000	🇬🇧
cause trouble	290,000	🇬🇧	cause results	260,000	🇬🇧
cause to your system	250,000	🇬🇧	cause a problem	230,000	🇬🇧
cause an increase	210,000	🇬🇧	cause changes	160,000	🇬🇧
cause the system	150,000	🇬🇧	cause serious problems	150,000	🇬🇧
cause side effects	140,000	🇬🇧	cause any problems	140,000	🇬🇧
cause some people	130,000	🇬🇧	cause different side effects	110,000	🇬🇧
cause a person	110,000	🇬🇧	cause different side	110,000	🇬🇧
cause a change	99,000	🇬🇧	cause unwanted effects	97,000	🇬🇧
cause a system	83,000	🇬🇧	cause loss of	80,000	🇬🇧
cause illness	75,000	🇬🇧	cause the actual results	75,000	🇬🇧
cause the body	75,000	🇬🇧	cause notice	75,000	🇬🇧
cause some problems	70,000	🇬🇧	cause side	69,000	🇬🇧

Figure 1.12: The results of searching for lexical form ***cause + Noun***.
(Image taken from Interactive language learning – flax library (2010))

From the list of results above, the user can see typical examples of a phrase by clicking on it. Figure 1.13 shows some examples when clicking on "cause problems".



Web examples

There are many things which can cause problems for your pet dog's skin; however, most of them can be managed by proper care by the owner.

Home FAQs Cracking & Joint Problems in Concrete My wife and I are building a new home. have noticed hairline cracks. Will this cause problems? Home

Cause Problems, Payroll People Inc of Payroll People Inc''s information - including email, business address, business phone, biography, title, company, jobs and ...

... Multiple \fromSlide* cause problems - ID: 424683 ... Details: I have the following pstricks code as part of a slide inside an overlays ...

mess sb up (CAUSE PROBLEMS) - definition, audio pronunciation, synonyms and more for mess sb up (CAUSE PROBLEMS): to cause someone to suffer emotional and mental ...

Figure 1.13: Example of some sentences using ***cause*** + ***noun.***
(Image taken from Interactive language learning – flax library (2010))

1.4.Motivation

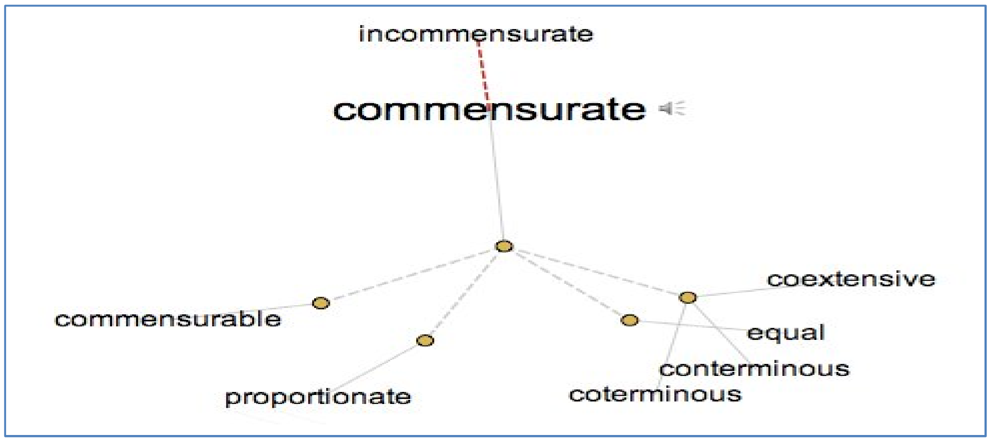
Like we mentioned above, the concordancing means in language study become popular. In terms of collocational study, web derived corpus developed by Wu, Witten and Franken (2010) not only provides massive results, but also utilizes the accessibility of the internet, making it very convenient for learners.

However, the current representation still contains several disadvantages. In detail, when users input a word, website returns a list of the word's lexical combinations, but each lexical form is explored separately in individual window. Hence, this approach prevents users to explore lexical forms as a whole unit, which may cause the users not able to observe "the whole picture" and sense a comparison between results from different lexical combinations.

Secondly, the current representation displays the results in descending order, along with their frequencies. This way helps the users to access top examples with a quick glance, but the difference in frequency should be encoded in lexical form's visual representation which will increase the importance of the top results and help users to gain more insight.

Moreover, it is clear for us that there is connection between any pair of information display; for example, a connection between a given word and its lexical forms, or a lexical form and its results. As a result of those observations, we can design a better visual representation for this dictionary by using a graph layout.

In fact, graph visualization has been using in some on-line dictionary software. Notable example is a graphical web-based thesaurus from Thinkmap Visual Thesaurus website (http://www.visualthesaurus.com/). Results are displayed as a tree-like structure with the input word as its root; its nodes are expanded with animations. This approach indeed raises users 'attention and awareness, which directly improve user's cognitive functions. In addition, the graph has elastic behavior in which the edges have a spring like behavior, and nodes automatically repel each other when being dragged too close to each other. This approach not only prevents visual clutter even if text which labels nodes is too long, but also supports flexibly local explorations such as clicking, or dragging each individual node (Figure 1.14)



.

Figure 1.14: An example of Visual Thesaurus dictionary
(Image taken from http://www.visualthesaurus.com/)

Another typical example we should mention is Visuwords online graphical online dictionary (http://www.visuwords.com/). The dictionary allows users to look up English words to find their meanings and associations with other words and concepts. It utilizes colors and shapes not only on nodes to indicate different categories of related words such as nouns, verbs or adjectives, but also on edges in order to indicate different relationships between the searched words with their associated words. The graph is also built to behave elastically. However, unlike Visual Thesaurus above in which clicking on a word will rebuild a new graph centered on it, Visuwords dictionary allows users to expand further sub nodes on the current graph.

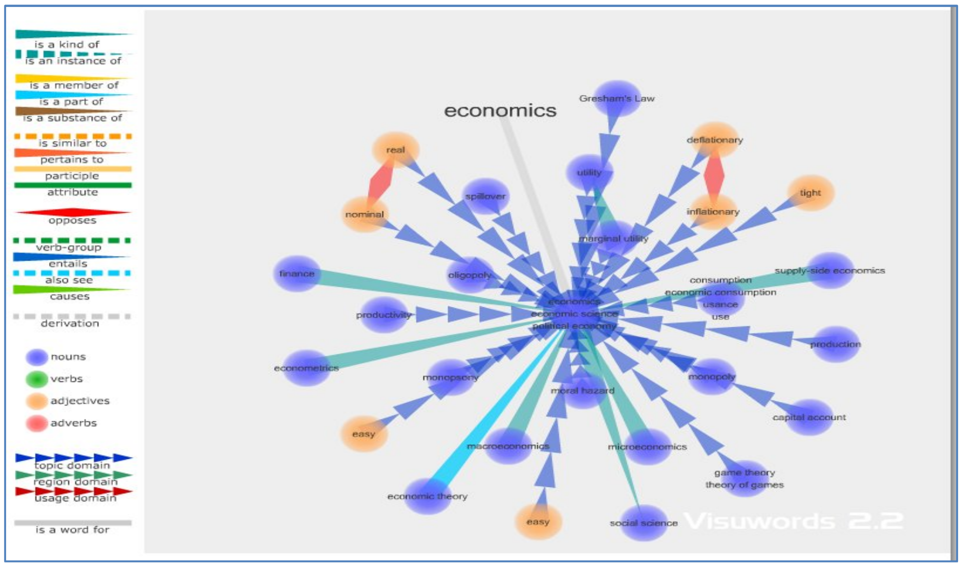


Figure 1.15: An example of Visuwords software
(Image taken from http://www.visuwords.com/)

According to Bederson & Boltman (1999), they pointed out that “animation facilitate learning” and with all the advantages of the graph layout mentioned above, we can clearly recognize that visual graph representation is more suitable than conventional one in language study. As a result of that, our attempt to apply graph layout into visualizing collocational search is reasonable, and a good outcome is promising.

Chapter 2 –Literature

2. Related work

2.1. Graph drawing

Graph layout is one of the most vital elements of graph visualization applications. There is a research area in which they study how to draw a nice graph is called “graph drawing”.

In fact, the procedures for drawing a graph can be explained simply as follows: given a set of nodes with a set of edges, designers will calculate the position of the nodes and connect all edges to their correct nodes. Drawing a graph is simple, but drawing a graph in which it respects some popular aesthetic criteria such as symmetry or no edge crossings requires an understanding of the input graph’s structure in advance (Battista, Eades, Tamassia and Tollis, 1999). In fact, a graph which satisfies the condition “no edge crossing” is called planar graphs. Drawing a planar graph can be simply achieved by introducing dummy nodes at the edge crossings (Kobourov, 2004). Planar graphs have a very pleasing visualization, but they hardly are used in real visual applications (Johannes, 2005).

In fact, aesthetic qualities improve the way that humans perceive graphs (Hu, 2005). In addition, aesthetic criteria are one of the main inspiration which drives research in graph drawing (Huang, Eades, Hong and Lin, 2010; Lin & Yen, 2011). In detail, the most important criteria are mentioned as follows:

- Minimize number of edge crossings: edge crossings affects keeping track of all connections.
- Minimize total drawing area: inefficient use of drawing area affects reading graphs.
- Maximize symmetry: graphs with high symmetry are more readable to humans. In terms of symmetry, “preventing zero angular resolution” (preventing two edges starting from the same node to overlap each other) is also an important criterion.

However, it is hard to develop an algorithm that satisfies all these aesthetic criteria. Huang, Eades, Hong and Lin (2010) pointed out that achieving one aesthetic criterion may cost another one. An example they showed maximizes degree of symmetry, but also allows large number of edge crossings which violates "minimize number of edge crossings" criterion (Figure 2.1).

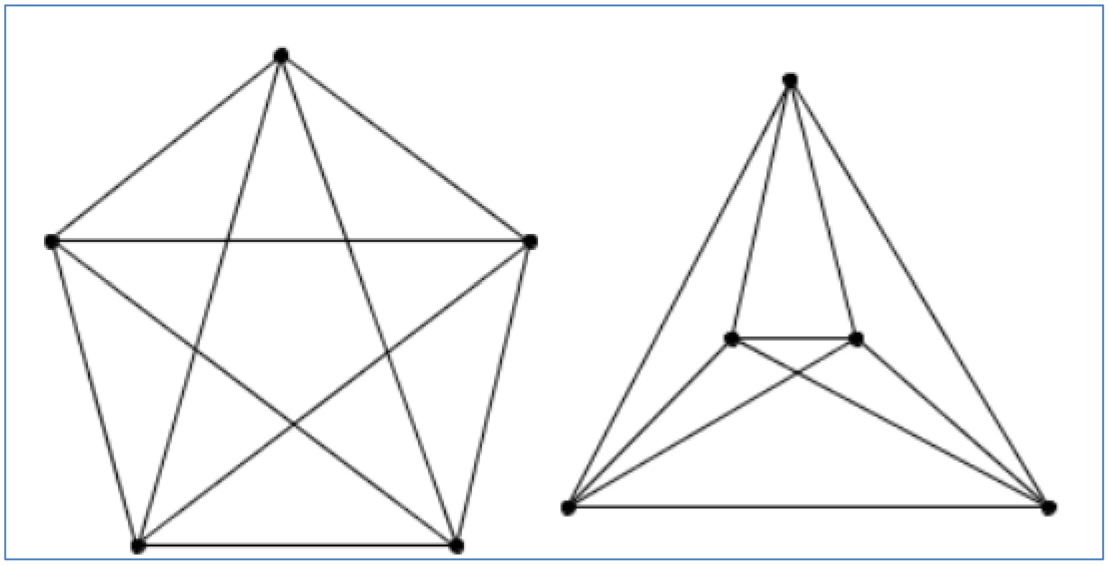


Figure 2.1: The same graph with different layouts
Left image: It has high degree symmetry, but violates "minimize number of edge crossing"
Right image: It achieves "minimize number of edge crossing", but does not have nice symmetry
(Picture taken from Huang, Eades, Hong, and Lin, 2010).

Until now, there have been quite a number of graph layout algorithms and they are classified based on what types of graphs to which they can be applied. For general graphs, force-directed algorithms are the most popular because it is easy to implement and designers do not need to know the input graph's structure and deep knowledge of graph theory (Johannes, 2005).

2.2. Force directed graph algorithm

a) Introduction

Force-directed algorithm is also called as spring-embedded model. It considers a graph as physical bodies tied with springs. Graphs drawn with these algorithms tend to be "aesthetically pleasing, exhibit symmetries and tend to procedure crossing-free layouts for planar graphs" (Kobourov, 2004).

b) Eades' spring model

According to Kobourov (2004), force-directed graph algorithms have been appeared since early 1963. The spring model came out in 1984 by Eades (1984), who considered a graph's components as mechanical objects. He described his model that "we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system". Eades (1984) explained the way it works that "the vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state". In detail, after letting the system go, there are two forces appeared— an attractive force between any two connected nodes and a repulsive force between any pair of nodes in the system. Particularly, Eades (1984) developed the attractive force based on Hookes Law formula but he suggested using the logarithmic form because his experiments showed that Hookes Law linear springs were too strong when the vertices were far apart.

Hookes Law formula:

$$\boxed{F_a = k*(d-L)} \qquad \text{(Formula 2.1)}$$

In formula 2.1, d is the current length of the spring (or distance of the two connected vertices), k is the stiffness constant and L is default spring length.

Eades logarithmic spring formula:

$$\boxed{F_a = c_1 * \log(d/c_2)} \qquad \text{(Formula 2.2)}$$

In formula 2.2, d is the current length of the spring (or distance of the two connected vertices), c_1 and c_2 are constant ($c_1 = 2$ and $c_2 = 1$ in practice).

In addition, the repulsive force is calculated as follows:

$$\boxed{F_r = c_3/\sqrt{d}} \qquad \text{(Formula 2.3)}$$

In formula 2.3, d is the distance between two connected nodes and c_3 is the constant (in practice, $c_3 = 1$).

Therefore, given a graph G= (V, E), the combined force applied on vertex v is:

$$\boxed{F(v) = \sum_{(u,v)\in E} F_{a,uv} + \sum_{(u,v)\in V\times V} F_{r,uv}} \qquad \text{(Formula 2.4)}$$

In formula 2.4, $F_{a,uv}$ denotes the spring force and $F_{r,uv}$ denotes the repulsive force.

After initializing layout for all vertices, their positions will be adjusted by spring forces and vertex-vertex repulsive forces with each simulation step. The completed algorithm can be summarized in algorithm 1.

```
Algorithm 1: Eades's force directed model
-----------------------------------------
Input: a set of vertices V and set of edges E

Output: a force directed graph
-----------------------------------------
1.  assign initial locations of vertices in V
2.  loop M times   //according to Eades, M = 100

      //calculate repulsive force between nodes
3.    for v in V do begin
        // v.tempForce: temporary force of vertex v
4.      v.tempForce :=0   //reset all forces
5.      for u in V do
6.        if (u # v) then begin
            //Δ: the difference of positions of v and u vertices
7.          Δ:= v.pos – u.pos;
8.          v.tempForce:= v.tempForce + (Δ/|Δ|)*$F_r$ (Δ);
9.      end
10.   end

      //calculate attractive force
11.   for e in E do begin
        //e.v and e.u are the two vertices connected by e
12.     Δ:= e.v.pos – e.u.pos;
13.     e.v.tempForce := e.v.tempForce -  (Δ/|Δ|)*$F_a$ (Δ);
14.     e.u.tempForce := e.u.tempForce +  (Δ/|Δ|)*$F_a$ (Δ);
15.   end

      //update position for all vertices
16.   for v in V do begin
17.     v.pos = v.pos + v.tempForce;
18.   end
19. end
```

Algorithm 1: Eades's force directed model

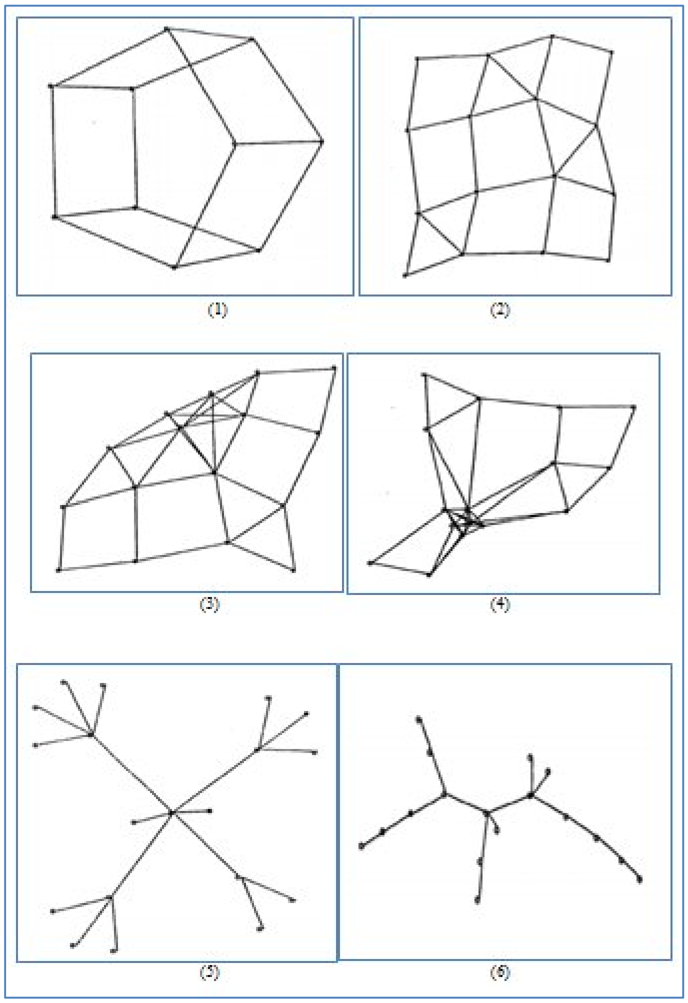


Figure 2.2: Graphs drawn by Eades model
(1), (2) Graphs with high degree of symmetry.
(3), (4) Graphs with tricky edge crossings.
(5), (6) Graphs with tree-like structure
(Image taken from Eades (1984))

Using this algorithm, the running time of calculating the attractive forces between adjacent vertices is O (E) and the running time of calculating the repulsive forces amongst vertices is O (V^2). Because O (V^2) is much greater than O (E) for most general graphs, the overall running time is O (V^2).

The Eades algorithm is indeed simple but it is effective. The produced graph achieves aesthetically pleasing and symmetrical geometry due to using the same edge length and balance of the two forces. However, this algorithm is only work well when the number of nodes is smaller than 30 (Kobourov, 2004); in fact, it still produces a number of edge crossings, but all vertices are distributed evenly.

c) Fruchterman and Reingold spring model (FR model)

Fruchterman and Reingold (1991) proposed an improved variant on Eades' algorithm. They also used spring embedded method; however, their formulas were quite different. Particularly, higher degree of the nodes' distance is used compared to the one in Eades' model. In Fruchterman and Reingold's model, there are also two forces; the repulsive force, F_r, appears between any two vertices, is inversely proportional to the distance between them. The attractive force, F_a, also exists between adjacent vertices and is proportional to the square of the distance.

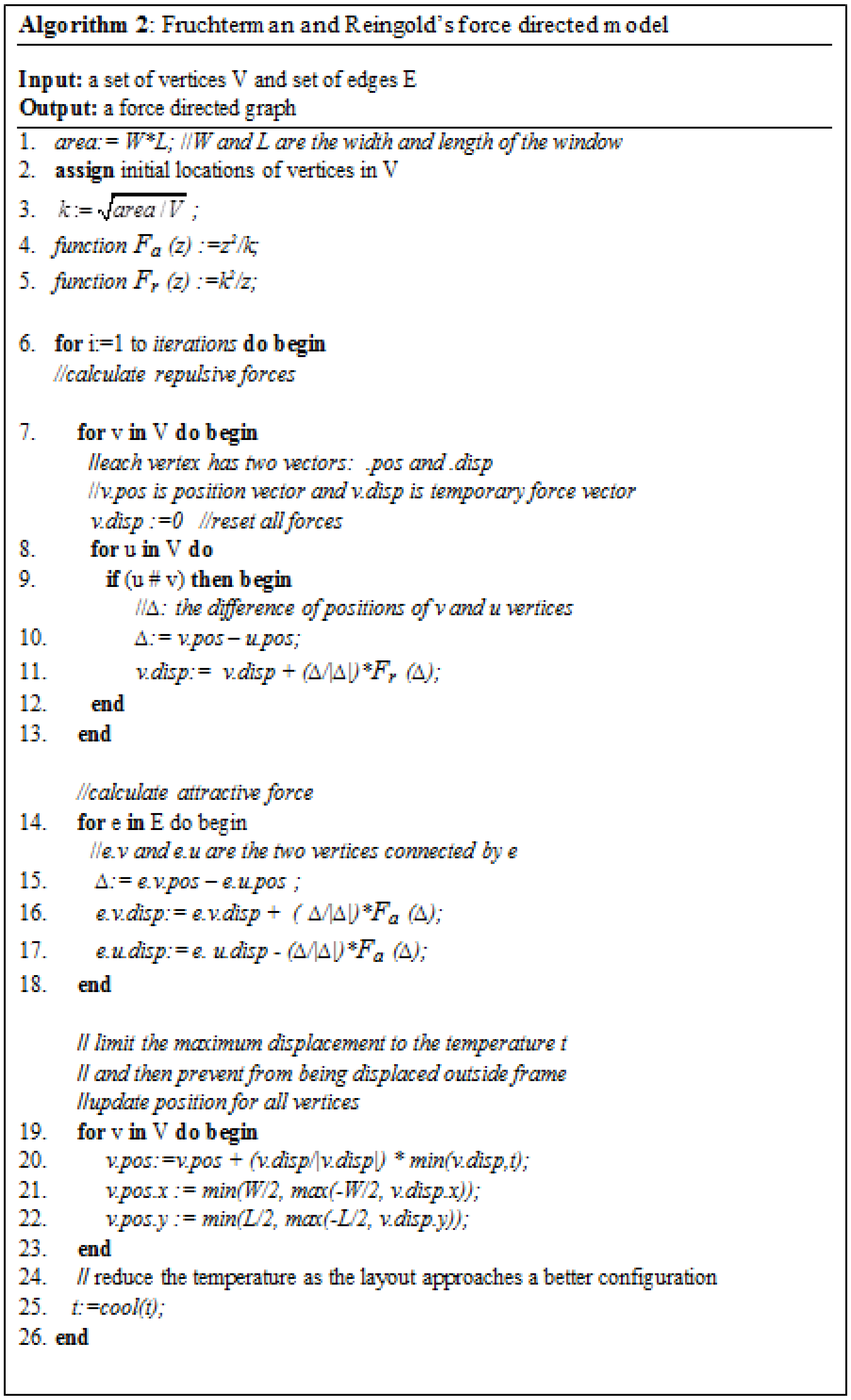
The two formulas of the forces

$$\boxed{F_r = -k^2 / d} \qquad \boxed{F_a = d^2 / k} \qquad \text{(Formula 2.4)}$$

In formulas 2.4, d is the distance between two vertices and k is called optimal distance between vertices. Its value particularly affects the size of graph.

$$\boxed{k = C * \sqrt{\frac{area}{numberOfVertices}}} \text{ (With C is a constant)} \qquad \text{(Formula 2.5)}$$

From the formula (2.4), it is easy to see that the both force relate strongly to the distance, leading to a problem of instability which is the forces exert on one vertex in the current simulation step can be very different in the next step due to changing positions of other vertices. The instability can result in not achieving minimal energy state for the whole graph system. Therefore, in order to deal with this problem, Reingold introduces two terms "temperature" and "cooling" process. Their method simply limits the displacement of a vertex to a maximum value and this value decreases over time by using the cooling function at each simulation step; so as the layout becomes better, the amount of adjustment becomes finer. The detail of algorithm is provided as follows:



Algorithm 2: Fruchterman and Reingold's force directed model

Input: a set of vertices V and set of edges E
Output: a force directed graph

```
1.  area:= W*L; //W and L are the width and length of the window
2.  assign initial locations of vertices in V
3.  $k := \sqrt{area / V}$ ;
4.  function $F_a(z) := z^2/k$;
5.  function $F_r(z) := k^2/z$;

6.  for i:=1 to iterations do begin
    //calculate repulsive forces

7.      for v in V do begin
         //each vertex has two vectors: .pos and .disp
         //v.pos is position vector and v.disp is temporary force vector
         v.disp :=0   //reset all forces
8.       for u in V do
9.         if (u # v) then begin
             //Δ: the difference of positions of v and u vertices
10.          Δ:= v.pos – u.pos;
11.          v.disp:=  v.disp + (Δ/|Δ|)*$F_r$ (Δ);
12.      end
13.     end

        //calculate attractive force
14.     for e in E do begin
         //e.v and e.u are the two vertices connected by e
15.      Δ:= e.v.pos – e.u.pos ;
16.      e.v.disp:= e.v.disp +  ( Δ/|Δ|)*$F_a$ (Δ);
17.      e.u.disp:= e. u.disp - (Δ/|Δ|)*$F_a$ (Δ);
18.     end

        // limit the maximum displacement to the temperature t
        // and then prevent from being displaced outside frame
        //update position for all vertices
19.     for v in V do begin
20.         v.pos:=v.pos + (v.disp/|v.disp|) * min(v.disp,t);
21.         v.pos.x := min(W/2, max(-W/2, v.disp.x));
22.         v.pos.y := min(L/2, max(-L/2, v.disp.y));
23.     end
24.     // reduce the temperature as the layout approaches a better configuration
25.    t:=cool(t);
26. end
```

Algorithm 2: Fruchterman and Reingold's force-directed algorithm

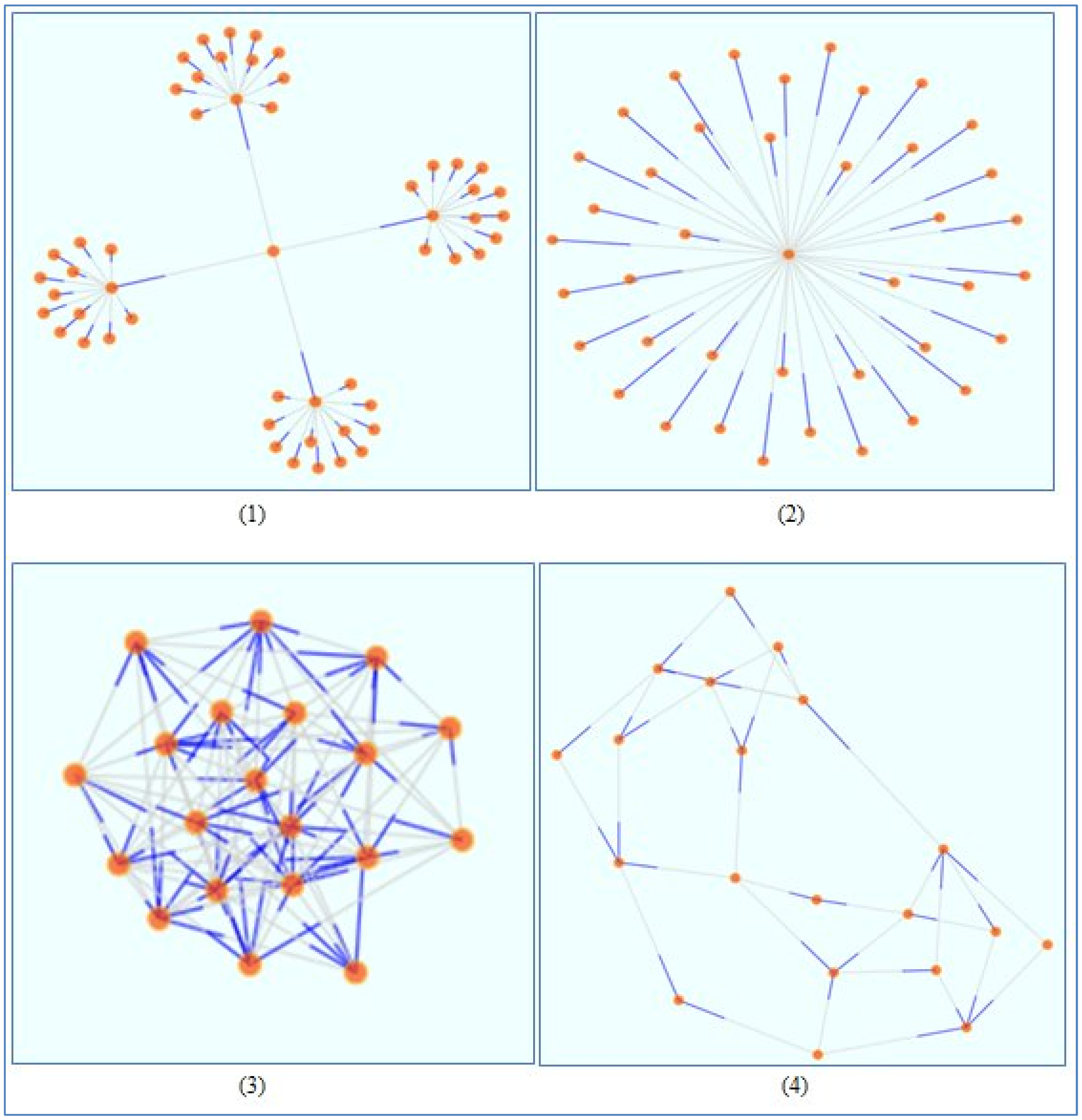


Figure 2.3: Examples of using Fruchterman and Reingold's algorithm.
(1): A graph with tree-like structure.
(2): a graph with its root branching off into 20 sub nodes.
(3): a dense graph (strong connected). (4): a sparse graph

In fact, Fruchterman and Reingold's algorithm has been used widely in research. As a result of this, its advantages and disadvantages have been examined thoroughly. In detail, compared to Eades's algorithm the number of edge crossings is smaller; moreover, this model indeed produces better vertex distribution (Fruchterman and Reingold, 1991). However, experiments show that it still have some layout's problem. According to Hu (2005), one drawback feature of a graph drawn by this algorithm is that vertices in the peripheral tend to be closer to each other than those in the center; Hu (2005) also called that was “peripheral problem” (Figure 2.4). Moreover, Yuan, Dancheng, Chunyan and Zhiliang (2009) also reported this problem in their experiment in which multi graphs drawn on the same layout were affected (Figure 2.5 below). As a result of this, the small graphs were pushed to the borders and corners.

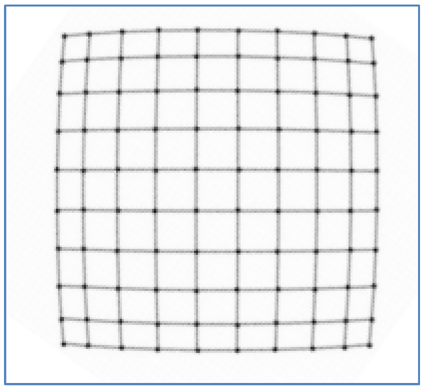


Figure 2.4: Peripheral problem in a graph with uniform mesh structure.
(Image taken from Yu (2005))

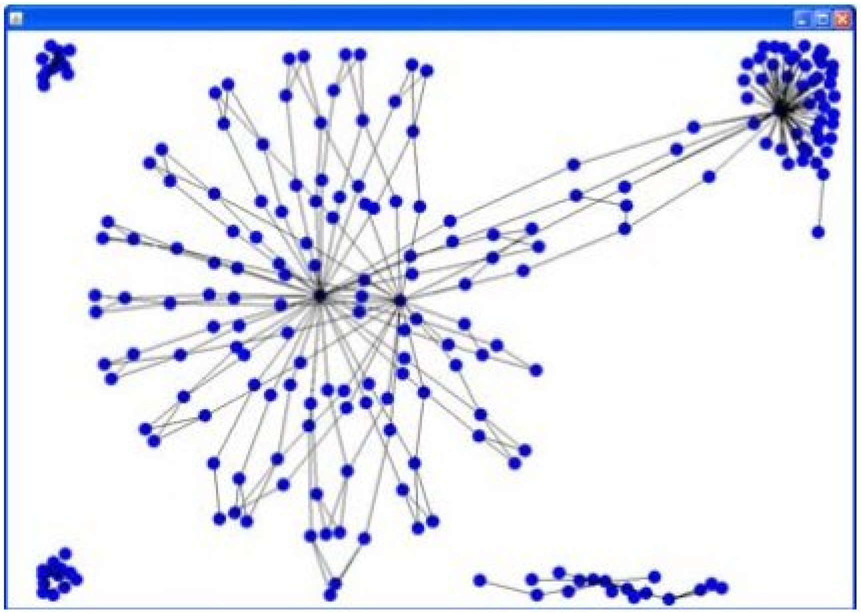


Figure 2.5: Peripheral problem in a multi graphs experiment.
(Image taken from Yuan, Dancheng, Chunyan and Zhiliang (2009))

d) Edge-edge repulsion model

Lin & Yen (2011) also proposed a new idea based on Eades's algorithm. Like we mentioned that aesthetic criteria drive the research in force-directed algorithm. Lin and Yen aimed to maximize degree of symmetry in graph drawing. Particularly, they aimed to maximize "angular resolution" which is defined as the smallest angle formed by two neighboring edges incident to the common vertex in a straight line drawing. In fact, achieving large angular resolution is very important in straight line drawings of huge graphs because it helps reducing edge-overlapping and visual clutter, improving graph clarification (Lin and Yen, 2011).

Their algorithm contained a lot of changes from Eades's. Firstly, an important change was that they used a final layout of a previous force-directed graph algorithm as an initial layout for their algorithm; therefore, their algorithm consisted of two phases. In detail, the first phase was to run classic algorithm (FR algorithm) from input vertices with random position; the second phase was to use the final layout it produced and then continue to run their algorithm in order to achieve a better layout.

Secondly, they use edge-edge repulsion instead of vertex-vertex repulsion. As a result of this, they can prevent "zero angular resolution" which appeared in some layouts created by previous algorithms.

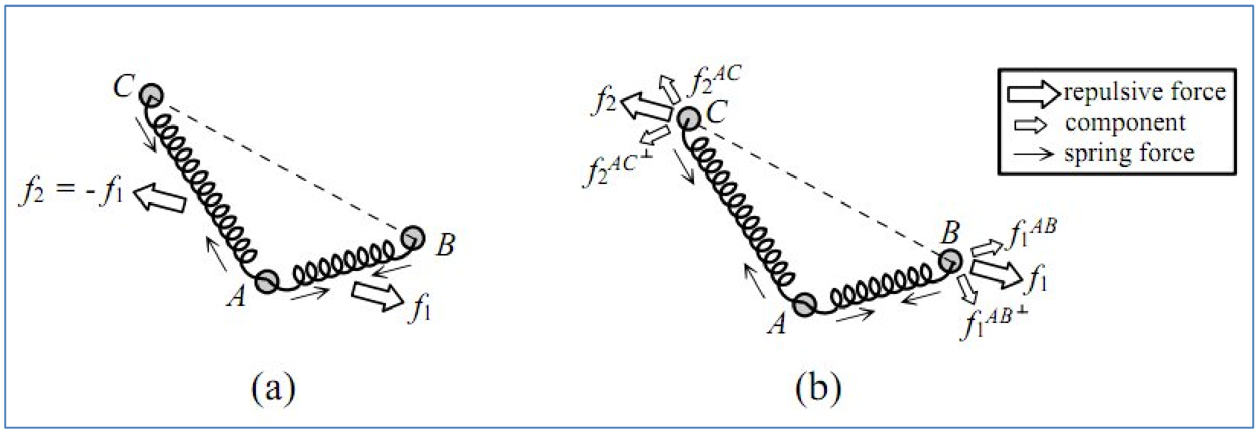


Figure 2.6: The edge-edge repulsion model.

(a) The force model where f_1 and f_2 are repulsive forces acting on $\overline{AB}$ and $\overline{AC}$.

(b) The position acted by repulsive forces f_1 and f_2 should be set at the end points B and C of incident edges of vertex. (Image taken from Lin and Yen (2011))

Lin and Yen (2011) stated that "the key in their edge-edge repulsion model is to express the repulsive force between two neighboring edges solely in terms of the lengths of the two edges and the included angle between the two edges". In detail, the authors considered that the magnitudes of the repulsive force between $\overline{AB}$ and $\overline{AC}$ are positively correlated with the length of $\overline{AB}$ and $\overline{AC}$, negatively correlated with the angle between $\overline{AB}$ and $\overline{AC}$ (angle θ).

The repulsive force is represented simply by the formula below.

$$\boxed{|f| = |f|_e + |f|_\theta} \qquad \text{(Formula 2.5)}$$

In formula 2.5, f is the repulsive force, f_e is the magnitude component and f_θ is the angle component. In addition, the total repulsive force f is applied on the ending nodes (Figure 2.6)

In detail, when constructing a formula for the magnitude component (f_e), they claimed that "the relationship between the magnitude of force and the total length of edges is non-decreasing and concave. Particularly, the magnitude should approach to zero as edge lengths approach to zero, and flatten out as edges lengths approach to infinity". This trend reflects the behavior of an arctangent function on (0, ∞). As a result of this, the magnitude component (f_e) can be simplified as follows:

$$\boxed{|f|_e = c_3 * (\tan^{-1}(\frac{|\overline{AB}|}{c_4}) + \tan^{-1}(\frac{|\overline{AC}|}{c_4}))} \qquad \text{(Formula 2.6)}$$

In formula 2.6, $\overline{AB}$ and $\overline{AC}$ are the lengths of edge AB and AC, respectively; c3 and c4 are constant to control the height of the approaching horizontal line and the scale of the horizontal axis.

In terms of angle component, Lin and Yen (2011) also observed the relationship between the angle included by AB and AC and the magnitude of repulsive force. It turned out that the curve is positive, non-increasing and convex. They reported that "the magnitude approaches to infinity as the included angle approaches to zero. On the other hand, the magnitude slowly flattens out as the include angle grows". As a result of this, a cotangent function on (0, π/2] was the suitable choice to reflect the relationship, and hence the amount that the component f_θ contributed to the magnitude of the repulsive force could be displayed as follows:

$$\boxed{|f|_\theta = c_5 * (\cot(\frac{\theta}{2}))} \qquad \text{(Formula 2.7)}$$

In formula 2.7, c_5 is a constant to control the scale of the vertical axis, and θ is the angle included by $\overline{AB}$ and $\overline{AC}$.

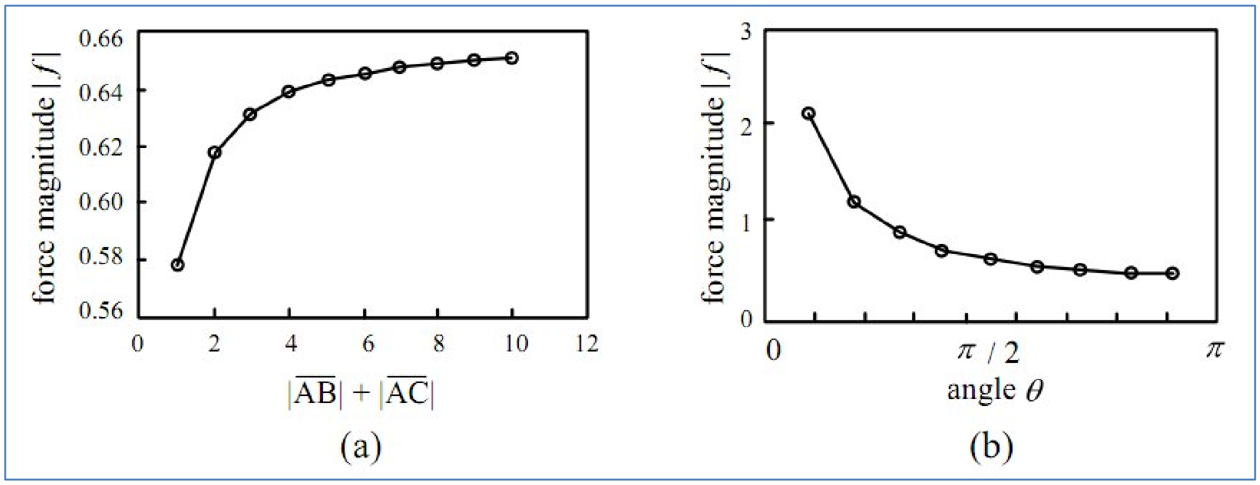


Figure 2.7: Relationship between f, the repulsive force and f_e, the magnitude component and f_θ, the angle. (a): Relationship between f and f_e. (b): Relationship between f and f_θ, the angle. (Image taken from Lin and Yen (2011))

However, Lin and Yen also calculated the attractive force between two connected nodes by their connected spring like Eades did. The formula used is presented as follows:

$$\boxed{f_a(d_{uv}) = (c_1 * \log(\frac{|d_{uv}|}{c_2}))\frac{d_{uv}}{|d_{uv}|}} \qquad \text{(Formula 2.8)}$$

In the formula 2.8, c_1, c_2 are constants and d_{uv} is the vector distance between two nodes u and v. Summary of Lin and Yen's algorithm can be represented as follows:

Algorithm 3: EERepulsion

Input: a reasonably nice drawing of G = (V, E)
Output: a nice drawing without zero angular resolution.
Require: *tempForce[|V|]* stores temporary forces of all vertices; *newPos* and *oldPos* record the new and old positions of all vertices, respectively.

```
1.  assign initial locations of vertices in V
2.  while converged ≠ 1 or the maximum iteration number is achieved do
3.     converged ← 1
4.     oldPos ← newPos
5.     initialize tempForce[|V|] as zeros matrix

6.     for each vertex v in V do
7.        if the degree of vertex v is at least two then
8.           for each  pair $(e_i, e_j)$ do
              // where edge $e_i = (v, v_i)$ and edge ej = $(v, v_j)$ are neighboring edges incident to v
              //calculate the repulsive force f according to formula 2.5, 2.6 and 2.7
9.              $tempForce[v_i] \leftarrow tempForce[v_i] + f$
10.             $tempForce[v_j] \leftarrow tempForce[v_j] - f$
11.          end for
12.       end if
13.    end for

14.    for each edge $e = (v, v_i)$ in E do
         //calculate the spring force $f_s$ of edge e according to formula 2.8
15.       $tempForce[v_i] \leftarrow tempForce[v_i] + f_s$
16.       $tempForce[v_j] \leftarrow tempForce[v_j] - f_s$
17.    end for

       //update newPos with tempForce
18.    for each vertex v in V do
         // $c_6(t)$ is a constant to control the magnitude of movement
19.      newPos ← newPos + min(tempForce[v], $c_6(t)$)
20.    end for
21.    if (||newPos − oldPos|| > ε then
22.       converged ← 0
23.    end if
24. end while
```

Algorithm 3: EERepulsion algorithm by Lin and Yen (2011)

Examples of using EERepulsion as the post processing in some graphs are represented in Figure 2.8 below. In detail, the classical algorithm used here is Fruchterman and Reingold's (FR algorithm). The "arrow sign" indicates that its target (right) takes its source (left) as the input layout. For example, EERepulsion either takes the layout of classical algorithm as its initial layouts in the first 3 graphs or the original initial layout for its input layout in the last graph.

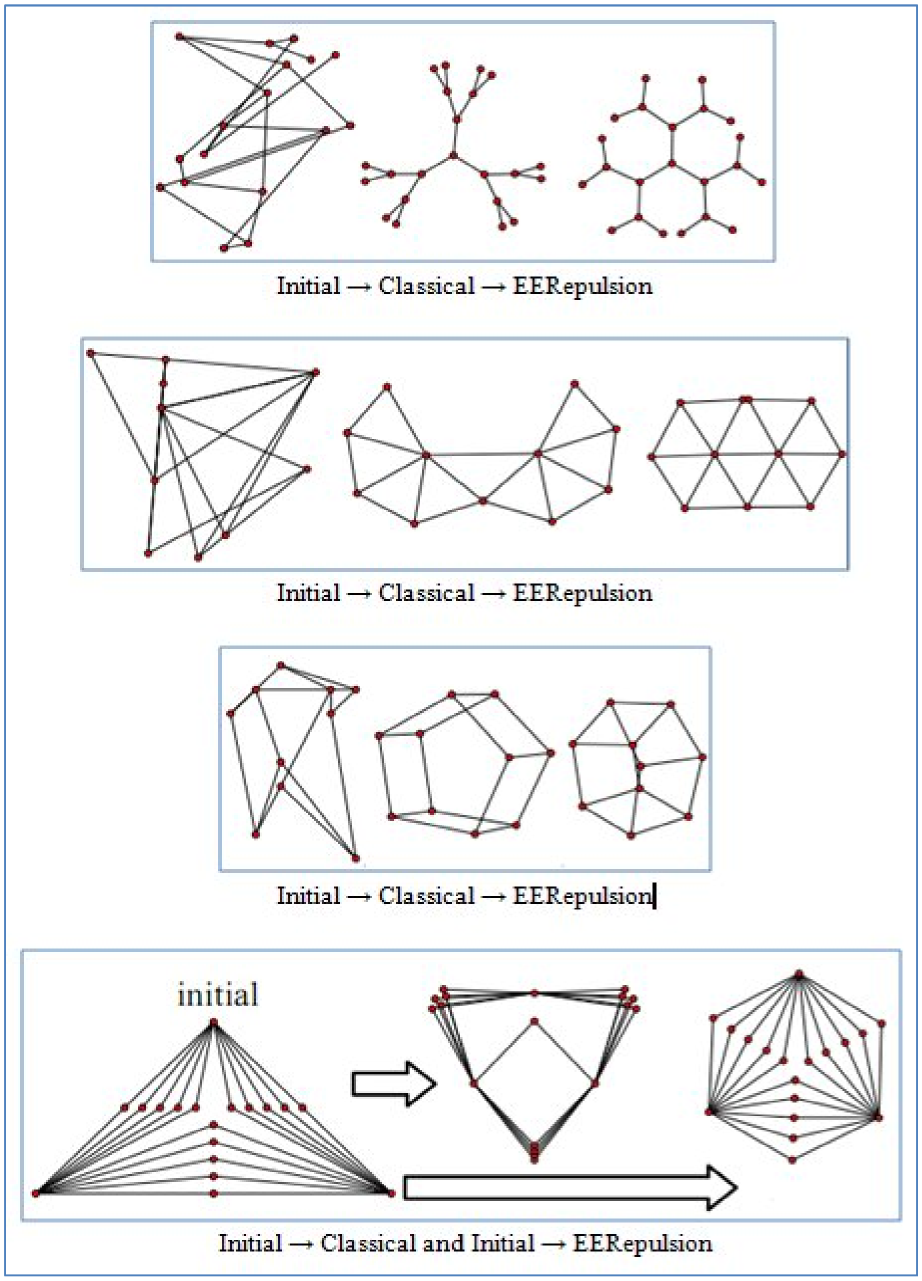

Initial → Classical → EERepulsion
Initial → Classical → EERepulsion
Initial → Classical → EERepulsion
initial
Initial → Classical and Initial → EERepulsion

Figure 2.8: Examples of using EERepulsion algorithm
(Picture taken from Lin and Yen (2011))

From Figure 2.8, we can easily that EERepulsion algorithm indeed creates optimal layouts for some particular graphs. For examples, graph 1 and 4 have high degree of symmetry, resulting in producing nice aesthetic pleasing. However, node-overlapping happens in the second graph and there is an area where nodes locates too close to each other (graph 3).

2.3. Methods to improve scalability

As we mentioned, force-directed graph is simple and convenient for general graphs. A lot of research have tried to improve aesthetic features for graph drawing; however, their algorithms still suffer the most weakness of force-directed model which is scalability.

In order to solve that problem, there have been some publications which can be categorized into two groups; firstly, they try to reduce the quadratic running time O (n^2) of calculating repulsive force by using optimized data structures which are called recursive space decompositions. Notable examples are quadtrees (Barn and Hut, 1986; Quigley and Eades, 2000) and recursive voronoi diagrams (Quigley and Eades, 2000; Pulo, 2001).

The second group is to design efficient ways to draw large graphs, which normally contain multiple steps of drawing. Simply, the large original graph is reduced to smaller versions which can recursively be reconstructed back the original. We start applying FR algorithm on the smallest version (coarsest version) in which the optimal layout can be easily achieved, and then step by step, the graph grows back to the original form. By using multi steps, the larger graph can easily make use of the optimal layout of the smaller graph in order to achieve its optimal layout. Notable examples are multi-scale algorithm of Hadany & Harel (2001) and multi-level algorithms of Walshaw (2003) and Yu (2005, 2011).

2.3.1. Recursive space decompositions (RSDs)

a) Introduction to RSDs

In force-directed algorithm, calculating repulsive force between all pairs of nodes is O (n^2), which is expensive when the number of nodes is large. In fact, this problem was considered as N-body problem in physics (Plafzner & Gibbon, 1996). When we consider repulsive force's formulas in all algorithms mentioned above, the repulsive force reduces in inverse proportion to the distance between two nodes. As the result of this, when we calculate repulsive forces applied on a node, other nodes which are considered to be far enough from it can be ignored. With this approximation, the quadratic running time can be reduced by using optimized data structures which are called recursive space decompositions.

According to Pulo (2001), "recursive space decompositions are spatial data structures which are defined recursively and at each level of the recursion divide space into smaller regions. Each region is then further subdivided at the next level of the data structure". It creates a tree structure in which the depth of nodes reflect the number of division from the root. The deeper a node is, the smaller region it represents. Depends on objects' size and position, they will be put in proper nodes. Moreover, objects which are near one another in space will also be located near one another in the RSD tree (Pulo, 2001).

b) Categorize RSDs

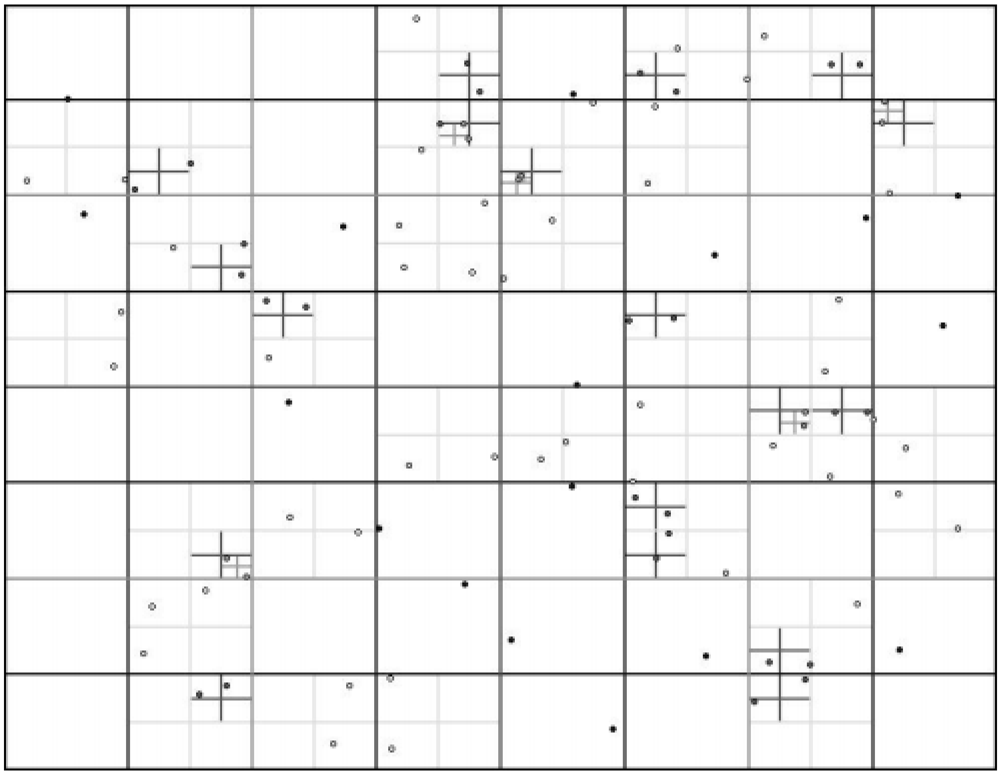


Figure 2.9: A quadtree of 100 points uniformly distributed in the plan (Limit 1 point per region). (Picture taken from Pulo (2001))

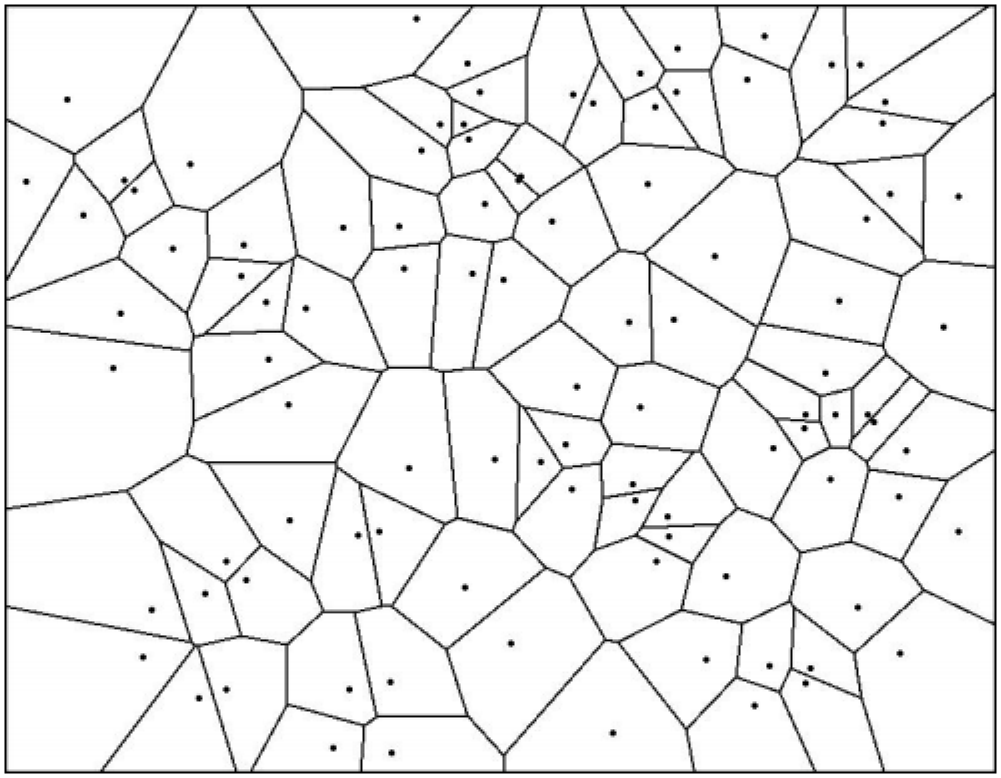


Figure 2.10: The Voronoi diagram of the 100 points in Figure 2.9.
(Picture taken from Pulo (2001))

RSDs can be divided into two broad categories: regular RSDs and irregular RSDs (Pulo, 2001). Regular RSDs divide space evenly at each level. Common examples are quadtrees. Irregular RSDs divide space into different sized and shaped regions at each level. Notable examples of this type are k-d trees and Voronoi diagrams (Quigley & Eades, 2000).

c) Introduction to Quadtrees

- Definition

In force-directed graph, the quadtree was first applied in Barn and Hut (1986). It is not only simple, but also provides effective performance. Tunkelang (1999) defined a quadtree as "a hierarchical partitioning of a rectangle, where each internal node of the tree represents a rectangle that has been split into four congruent sub-rectangles (bisected horizontally and vertically) and each leaf is an undivided rectangle".

- Create a quadtree

In theory, the tree has a limit of the number of particles stored at each node (bucket_max = 1). When we insert a particle, we begin at the root of the quadtree. Depends on the position of the input particle, it will be pushed into the proper node which represents the region where the particle locates. If the number of particles at the targeted node has not reached its limit (< bucket_max), we will add the input particle to this node. If the number of particles at this node is already at its limit (=bucket_max), we will branch this node into 4 sub-nodes, namely NW, NE, SE and SW. The particle is pushed down to one of the 4 sub-nodes. In addition, the targeted node also removes all its particles and passes them down to its 4 sub-nodes.

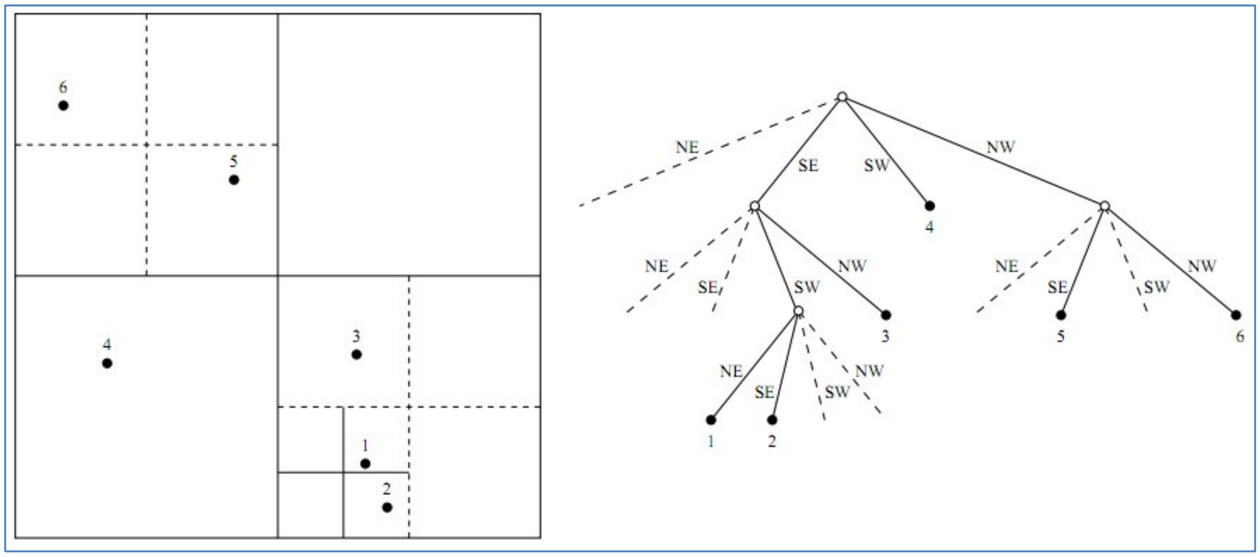


Figure 2.11: A simple quadtree (left) and its corresponding data structure (right)
(Image taken from Tunkelang (1999))

According to Tunkelang (1999), the time of inserting a particle is proportional to the height of the quadtree. Moreover, even if a particle is moved further down to a sub node, the total time spent on inserting that object is still proportional to its final distance counted from the root (Tunkelang, 1999). Therefore, the total time spent on creating the quadtree is proportional to the sum of the heights of its leaves. If the number of particles is N, then the lower bound of the depth of quadtree is Ω *(logN).* Creating a quadtree requires inserting N particles. Therefore, the lower bound of creating the tree is Ω *(NlogN).*

- <u>Query a quadtree</u>

An important advantage of using the quadtree is its optimal querying ability. For a given rectangle, the quadtree provides an optimized way for us to check how many cells the quadtree intersects. As we mention above that the internal node (internal node) represents a big rectangle which is divided into 4 smaller rectangles for the 4 sub nodes NW, NE, SE and SW. As a result of this, when we check whether the given rectangle intersect with nodes, if the input rectangle does not intersect with the parent node, then we can just ignore all its sub nodes. If they do collide, then we just recursively repeat the process with its sub nodes (Figure 2.12). Therefore, the lower bound running time of the query is **Ω** *(logN)*.

Moreover, when we are at the collided node, we can also check whether its particles intersect with the input rectangle. As a result of this, querying a list of particles which intersect with a given rectangle is fast. In fact, if we ignore far distant particles (particles locate outside the rectangle centered the focused particle) in calculating the repulsive force, the querying ability reduces the running time down to **Ω** *(NlogN)* (in the case that bucket_max = 1). In practice, people tend to set bucket_max = 4 and limit the depth of the quadtree to 10 in order to prevent repetition of removing and inserting objects at deep nodes.

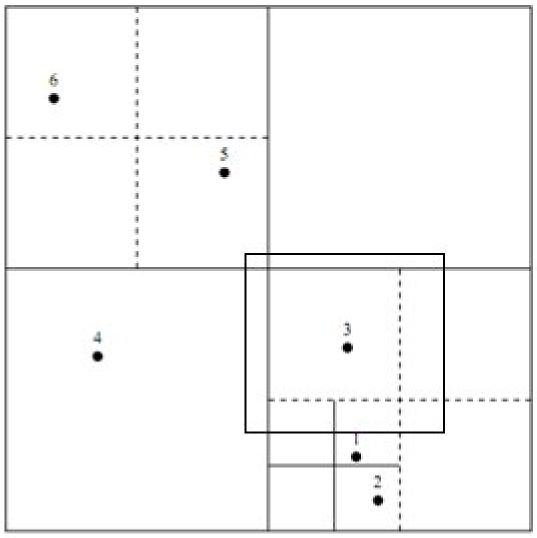


Figure 2.12: Visualizing a query in a quadtree.

To sum up, in order to visualize general graphs, the force-directed models are suitable choices because designers do not need to know much information about graph theory. The force-directed model describes a graph as a particle system and allows the repulsive forces between one particle and others. Furthermore, these particles are constrained by the attractive forces created from the spring embedded edges that they connect. In detail, by being affected by these forces, the particles adjust their positions step by step from random positions to form an optimal layout—the state of minimal energy.

The force-directed models tend to produce high degree of symmetry and aesthetic pleasing. In fact, they have been researched thoroughly since 1984; as a result of this, its scalable ability has been improved by using optimized data structure (Pulo, 2001), and multi-level methods (Hadany & Harel, 2001; Walshaw, 2003; Yu, 2005 and 2011).

Based on their benefits, it is clear that choosing a force-directed model for our visualization application is indeed a proper decision. However, choosing a suitable algorithm and implementing it are also challenging due to the fact that all force-directed algorithms are expected to use in off-line applications in which there is good hardware support and the entire graph information are known in advance. As a result of this, we come up with some modifications on the force-directed model and these modifications will be represented in the following chapter.

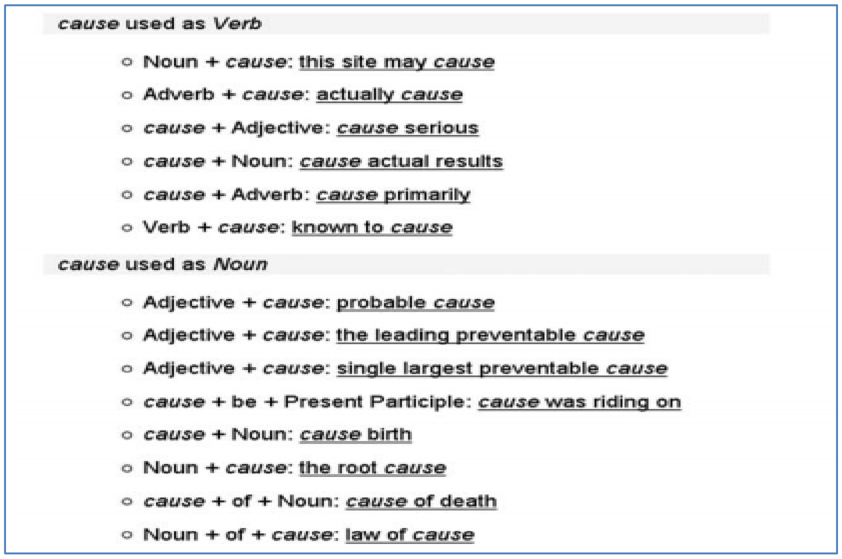
Chapter 3 – Implementation

3.1.Our approach to the design

Like we mentioned above, our attempt is to visualize web-derived corpus developed by Wu, Witten and Franken (2010) by using graph representations. The current representation of Wu is indeed simple and easy to use, but it can be optimized in order to facilitate learning and increase level of engagement, and these advantages can be achieved by utilizing graph representation in the visualization. However, when considering characteristics of the collocational corpus, the searching procedures, the advantages and disadvantages of the force-directed algorithms, designing efficient visualization for the online dictionary of collocation words can be challenging. In fact, some modifications can be necessary in order to design a good layout.

3.1.1. Understand the collocational corpus and user interactions

From Wu's interface, when we input a word such as "cause", the web guide the user by displaying possible searching types of this word: "cause used as noun" or "cause used as verb". In addition, all possibly proper lexical forms listed for the user to choose.



cause used as *Verb*

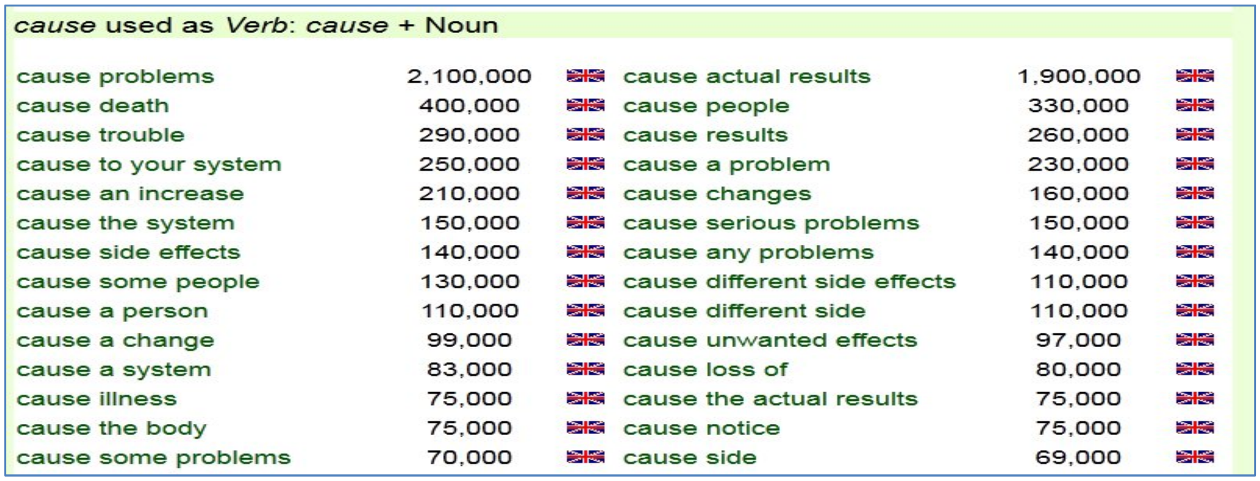
- Noun + *cause*: this site may *cause*
- Adverb + *cause*: actually *cause*
- *cause* + Adjective: *cause* serious
- *cause* + Noun: *cause* actual results
- *cause* + Adverb: *cause* primarily
- Verb + *cause*: known to *cause*

cause used as *Noun*

- Adjective + *cause*: probable *cause*
- Adjective + *cause*: the leading preventable *cause*
- Adjective + *cause*: single largest preventable *cause*
- *cause* + be + Present Participle: *cause* was riding on
- *cause* + Noun: *cause* birth
- Noun + *cause*: the root *cause*
- *cause* + of + Noun: *cause* of death
- Noun + of + *cause*: law of *cause*

Figure 3.1: Result of input word "cause".
(Image taken from Interactive language learning – flax library (2010))

Furthermore, when the user clicks on one of these lexical items, a list of results will pop out.



cause used as *Verb*: *cause* + Noun

cause problems	2,100,000	🇬🇧	cause actual results	1,900,000	🇬🇧
cause death	400,000	🇬🇧	cause people	330,000	🇬🇧
cause trouble	290,000	🇬🇧	cause results	260,000	🇬🇧
cause to your system	250,000	🇬🇧	cause a problem	230,000	🇬🇧
cause an increase	210,000	🇬🇧	cause changes	160,000	🇬🇧
cause the system	150,000	🇬🇧	cause serious problems	150,000	🇬🇧
cause side effects	140,000	🇬🇧	cause any problems	140,000	🇬🇧
cause some people	130,000	🇬🇧	cause different side effects	110,000	🇬🇧
cause a person	110,000	🇬🇧	cause different side	110,000	🇬🇧
cause a change	99,000	🇬🇧	cause unwanted effects	97,000	🇬🇧
cause a system	83,000	🇬🇧	cause loss of	80,000	🇬🇧
cause illness	75,000	🇬🇧	cause the actual results	75,000	🇬🇧
cause the body	75,000	🇬🇧	cause notice	75,000	🇬🇧
cause some problems	70,000	🇬🇧	cause side	69,000	🇬🇧

Figure 3.2: Top results of lexical form "cause + Noun" (the number of results > 50)
(Image taken from Interactive language learning – flax library (2010))

From these steps above, it is clear for us that the collocational database contains tree-like data in which each single word such as "cause, sun or take" is the root of its own tree. When we visualize the data as a graph, "the input word", its searching types and its lexical forms can be considered as backbone nodes and they will be shown at the beginning in order to help users navigating. Moreover, they can be animatedly displayed in order to help users to see "the flow" and raise their awareness.

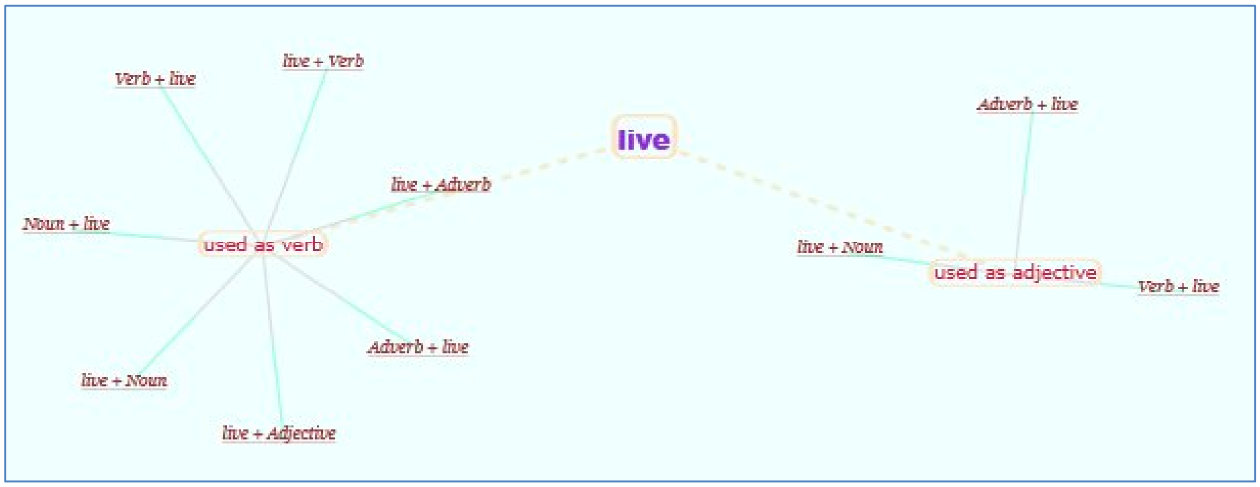


Figure 3.3: A simple design of the important nodes

As we mentioned above, knowing the structure of visualized data always works to designers' advantage and in this case, we have more benefits because the forced directed model of tree-like graph is proved to be stable.

Furthermore, from Figure 3.2, the popularity of the resulted phrase is indicated by its frequency. So visualizing efficient this feature in the graph layout is also important in order to help users to improve decision-making.

However, the database also has some specific characteristics which may cause problems if we want to use force-directed models (experiment with Eades' and FR models). For example, from Figure 3.2, it is clear for us to see that the number of searching results can be very large (>50) in some lexical combinations such as "word + noun" (cause + noun). So, with graph representation, the node labeled "cause + noun" will branch off into more than 50 child nodes in order to show the top results (Figure 3.4). Consequently, the constraint force from these spring embedded edges is large on the branching node. Whenever users interact with the branching node and this force causes the node to react very slowly when the users try to control. Therefore, some modifications on the force-directed models are necessary to solve this problem.

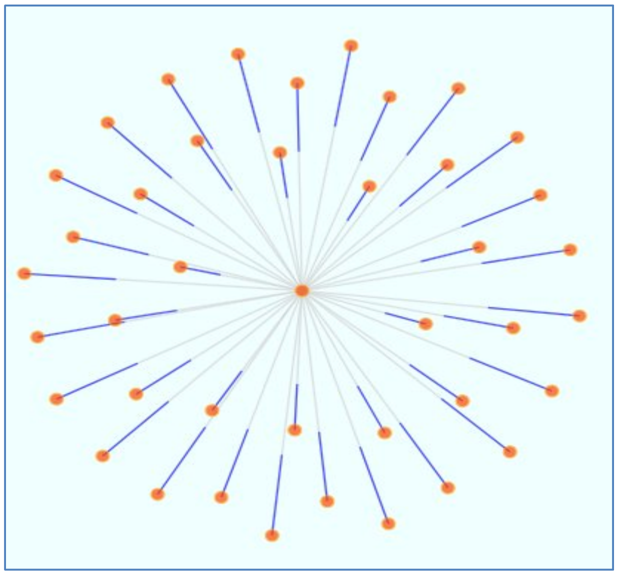


Figure 3.4: A graph example of one centered node with 35 nodes (Using FR algorithm)

Furthermore, Figure 3.2 shows that the resulted phrases are too long. Labeling the sub nodes with these phrases can cause visual clutters preventing the users from reading properly. Hence, finding a good way to display the resulted phrases also is challenging.

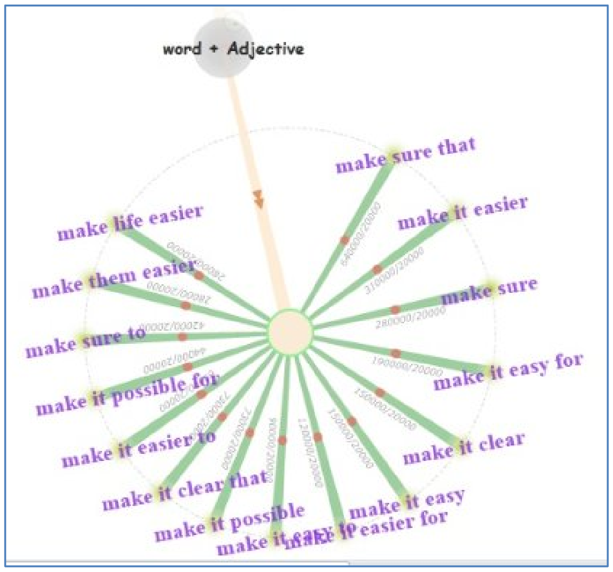


Figure 3.4b: An attempt of visualizing results of the lexical pattern (make +Adjective)

3.1.2. Understand the current force-directed algorithms

As we mentioned that the force-directed algorithms, especially FR algorithm, are suitable to display all general graphs. However, when we apply these models into information visualization applications, there are quite a number of drawbacks.

Firstly, the force-directed models such Eades's and FR models, have no control on some output features such as the edge length— the feature may be necessary to encode some attributes of displayed information.

Secondly, all these force-directed models have been designed for off-line visualization in which the entire graph information (vertices and edges) should be known in advance; for example, FR algorithm requires the number of vertices in advance in order to calculate the scale value: $\boxed{k = C * \sqrt{\frac{area}{numberOfVertices}}}$. Even though the structure of our graph is known, the information of vertices and edges is still unknown and the expansion of our graph depends on user's interactions. As a result of this, applying the original force-directed models is quite challenging.

Thirdly, when visualizing tree-like data such as our data, both Eades's and FR models produce a drawing in which the sub-nodes tend to be pulled away from the center root. So, the whole graph will require a lot space to be displayed (Figure 3.5). This space-consuming problem will be a major drawback if we plan to display searched results of more two words on the same screen. Particularly, the most popular algorithm, FR model, also suffers from the peripheral problem (Figure 3.5), which seriously prevents us from displaying properly results of multi words in the same layout.

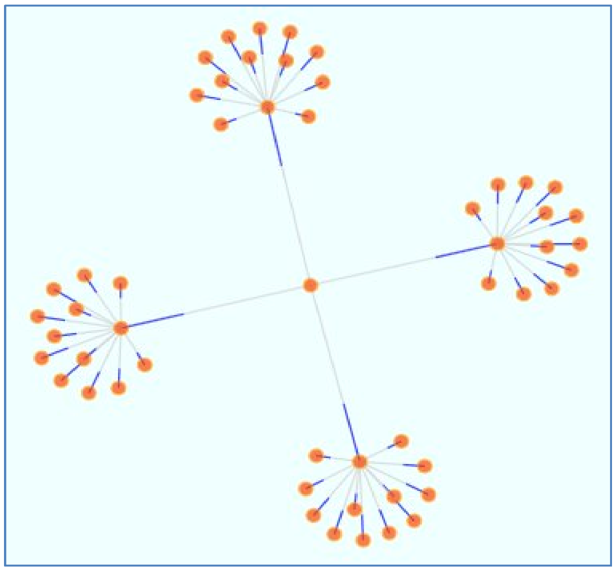


Figure 3.5: An example of tree-like data (FR algorithm)
(The sub nodes of each branch tend to be pulled close together)

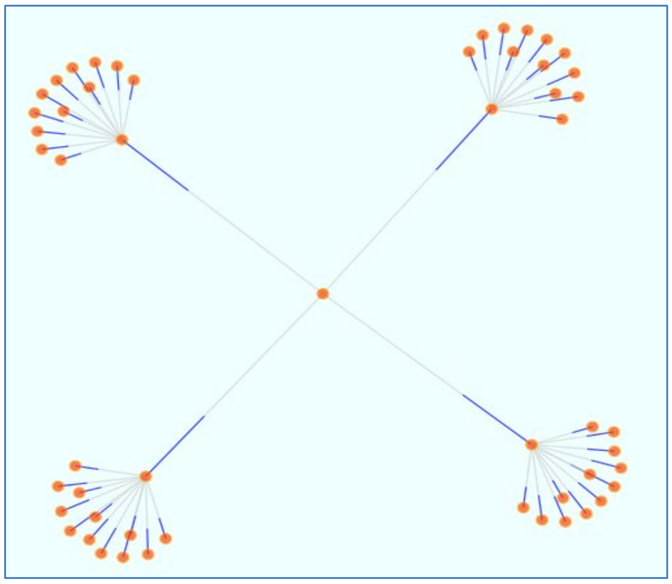


Figure 3.6: An example of tree-like data (Eades's algorithm)
(The sub nodes of each branch tend to be pulled away from the center)

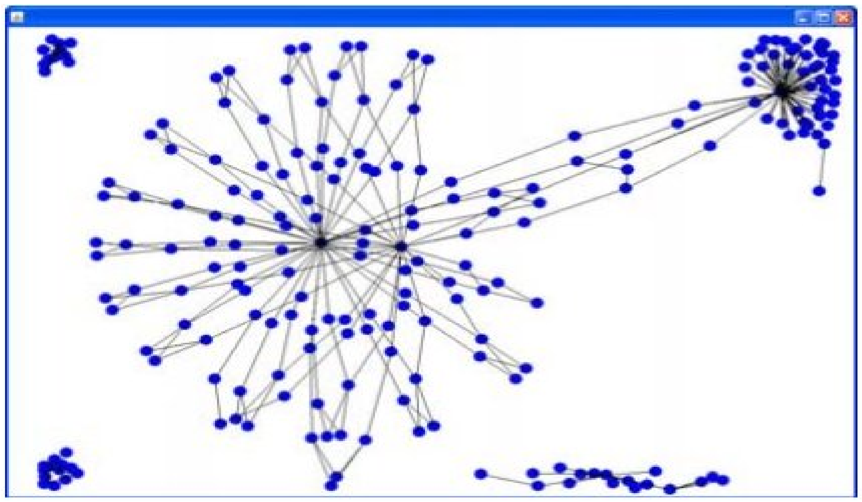


Figure 3.7: An example of visualizing multi graphs on the same screen with FR algorithm
(Image taken from Yuan, Dancheng, Chunyan and Zhiliang (2009))

Even though the EERepulsion model developed by Lin and Yen (2011) produces high degree of symmetry which reduces greatly visual clutters in sub-node areas, it is still not suitable for online drawing because of its complicated implementation (require two steps in drawing).

To sum up, the characteristics of the graph visualization application and the disadvantages of the current force-directed algorithms indeed pose some challenges. In order to create an effective application, we implement some modifications on the current force-directed model by combining Eades‘s and EERepulsion models and redesigning a new spring embedded edge model. In fact, these modifications will be represented in detail in the next section.

3.2. Our design and implementation

3.2.1. Design the force formulas

In order to obtain more controls on the output features of the force-directed model, we re-design the formulas of the repulsive and attractive forces. As we observed, Eades' and FR algorithms do not control the length of their edges – the feature which is very useful to encode information. We formulate the attractive force as follows:

The attractive force formula between two nodes connected by an edge:

$$\boxed{f_a = k * (L - d)} \qquad \text{(Formula 3.1)}$$

In formula 3.1, L is the default length for edges, k is a constant and d is the distance between the two connected nodes. It is clear that we re-use the Hook's law formula for our edges because it allows us to control the output length of the graph edges.

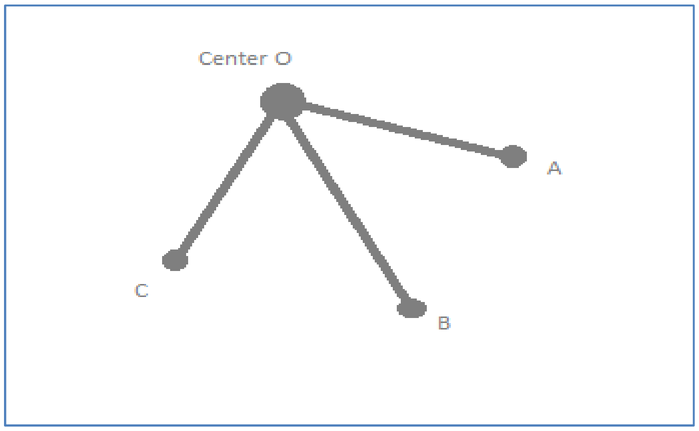


Figure 3.8: Hook's law on tree-like graph

However, when using our attractive force on tree-like data (Figure 3.8), the center node (node O) will be received large amount of force if its initial position is not optimal. So, it may cause this node to behave unstably. In order to solve this problem, we adjust the attractive force formula for different kind of nodes: branching nodes and ending nodes. In detail, the branching node (node O) should receive less force; on the other hand, the ending nodes like node A, or B should receive more attractive force.

. The adjustments for the attractive force model for branching nodes such as node O and ending nodes such as node A, B or C

$$\boxed{|f_{a,O}| = c_1 * |f_a|}$$ (Formula 3.2 a)

$$\boxed{|f_{a,A}| = |f_{a,B}| = |f_{a,C}| = c_2 * |f_a|}$$ (Formula 3.2 b)

In formula 3.2, c_1 is a constant ($c_1 = 0.25$ in our experiments). In formula 3.3, c_2 is also a constant ($c_2 = 4$ in our experiments).

As a result of this adjustment, when users move the center node, they also easily affect other sub nodes A, B and C. When they drag the ending node such as node A, the center node is hardly affected; therefore, the entire graph layout cannot be affected if sub nodes are interacted.

Furthermore, symmetry is also a criterion that we consider. If all sub nodes branched off from the same center are distributed evenly, they will make the graph more readable and allow us label on these nodes. Hence, in order to improve symmetry for our graph, we follow EERepulsion model (Lin & Yen, 2011) and add a repulsive force between every pair of edges having the same starting vertex. However, we also recognize that the lack of the repulsive force between every pair of vertices in EERepulsion model causes node overlapping problem in some particular graph layouts. Therefore, in our algorithm we also implement the vertex-vertex repulsive force which is similar to the one of Eades's.

The vertex-vertex repulsive force and the edge-edge repulsive force between two neighboring edges $e_1 = (o, u)$ and $e_2 = (o, v)$

$$\boxed{f_r = c_0 / \sqrt{d}}$$ (Formula 3.3)

$$\boxed{f_{u,v} = c_3 * \cot(\frac{\theta}{2})}$$ (Formula 3.4)

In formula 3.4, θ is the angle between the two edges. In addition, the force $f_{u,v}$ is applied onto the two ending nodes u and v.

3.2.2. Design data structure

Because we design an online-dictionary-software, reducing the complexity is indeed crucial. So, we use a quadtree as our data structure. However, we also did some modifications on building the quadtree in order to make it more suitable to the online visualization.

For off-line graph visualization, all the nodes are expected to be drawn on the drawing area at the same time. Hence, it is certain that they use the largest fixed region for the root tree node in order to perform inserting on other graph nodes. However, our case is quite different. As we explained, our graph only contains some navigating nodes at the beginning. Depends on how users interact, it will expand the branches that the users want. As a result of this, it is unwise if we reserve a big area for the root node. Moreover, it may complicate the inserting nodes into the quadtree in some particular situations in which the positions of nodes are not desirable (Figure 3.9)

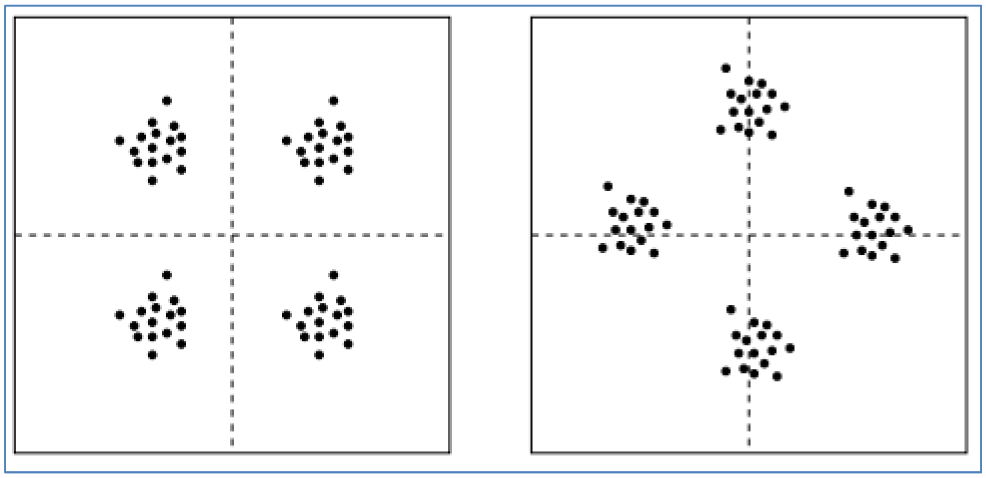


Figure 3.9: Examples of unfavorable nodes' position
(Image taken from Pulo (2001))

In detail, our approach is to provide the root node with the smallest area (the area that can't be divided). When a new node is inserted, the quadtree will check whether the root's area covers the new node or not. If the new node is outside this area, the quadtree will recursively expand the root's area to cover the new node (from Figure 3.10 to Figure 3.12).

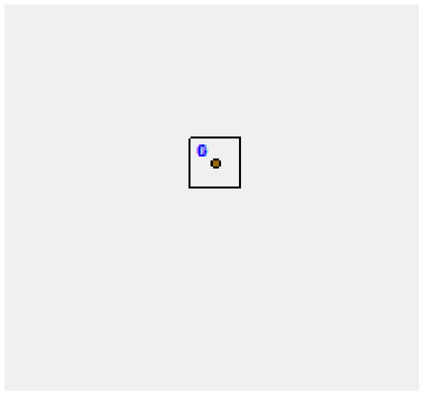


Figure 3.10: The root node with the smallest area (25, 25)

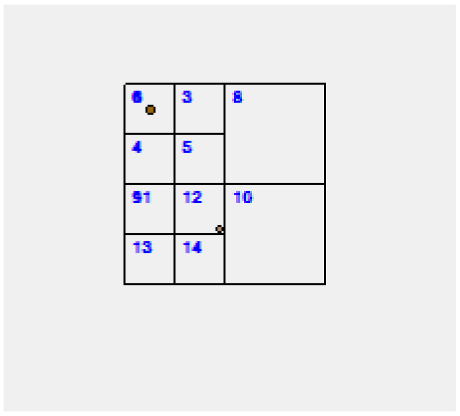


Figure 3.11: The root's area expands recursively to cover the new node

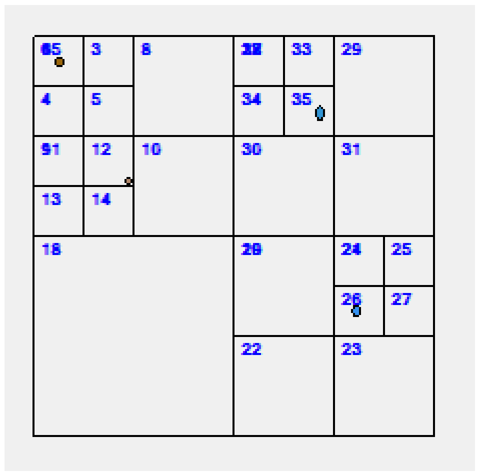


Figure 3.12: Further expansion of the root's area.

As a result of this modification, we only expand the calculating area when our graph needs and it is an efficient improvement especially for online drawing. In addition, it also reduces possible creating deep sub-nodes in the difficult cases (Figure 3.9).

3.2.3. Design a new curly spring embedded model

Like we mentioned above, for a given word, the number of resulted phrases of some particular lexical forms tends to be larger than 50. It leads to a case that there are some nodes which connect more than 50 sub nodes. This case is not suitable to be applied by the current force-directed algorithms because the amount of force exerts on these branching nodes is large and affects badly on user's interactions. In order to solve this problem, we design new curly spring embedded edge to replace the straight spring embedded edge which is used to connect between the branching node and its sub node. The new edge will reduce the amount of force applied on the branching node and make user's interactions more smoothly.

In detail, we can consider the new curly spring embedded edge as a series of 2 or 3 straight spring edges which are connected from one to another (Figure 3.10).

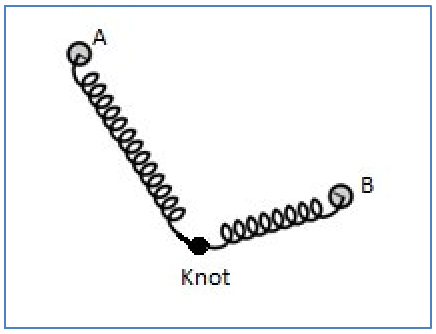


Figure 3.13: The curly spring embedded edge with two internal straight spring edges

From Figure 3.13, two nodes A and B are connected by 2 springs which have the knot (O) as their joint. In our design, the knot O is an internal node, so we do not include it in calculating the repulsive force amongst nodes like normal nodes A, or B. Moreover, because of the design, the attractive force exerts on each node is calculated differently.

If we consider the positions of node A, node B and node O are (x_A, y_A), (x_B, y_B), and (x_O, y_O). The distances between node A and knot O ($d_{A,O}$) and between knot O and node B ($d_{B,O}$) are calculated as follows:

$$\boxed{d_{A,O} = \sqrt{(x_A - x_O)^2 + (y_A - y_O)^2}} \qquad \text{(Formula 3.5)}$$

$$\boxed{d_{O,B} = \sqrt{(x_B - x_O)^2 + (y_B - y_O)^2}} \qquad \text{(Formula 3.6)}$$

With L is the default length of the spring and k is the stiffness, the attractive forces created by the two straight spring embedded edges can be calculated as follows:

$$\boxed{f_{A,O} = k * (L - d_{A,O})} \qquad \text{(Formula 3.7)}$$

$$\boxed{f_{O,B} = k * (L - d_{O,B})} \qquad \text{(Formula 3.8)}$$

If we consider that node A is a starting node and node B is an ending node, the attractive forces are represented in both x and y components in formula 3.9 and 3.10.

$$\boxed{F_{A_y} = \frac{(y_O - y_A)}{d_{A,O}} * c_1 * f_{A,O}} \quad \boxed{F_{A_x} = \frac{(x_O - x_A)}{d_{A,O}} * c_1 * f_{A,O}} \qquad \text{(Formula 3.9)}$$

$$\boxed{F_{B_y} = \frac{(y_O - y_B)}{d_{O,B}} * c_2 * f_{O,B}} \quad \boxed{F_{B_x} = \frac{(x_O - x_B)}{d_{O,B}} * c_2 * f_{O,B}} \qquad \text{(Formula 3.10)}$$

In formula 3.9, c_1 is a constant for starting nodes, c_1= 0.25 as the default value. In formula 3.10, c_2 is a constant for ending nodes, c_2= 4. As we mentioned, we use these values in order to make node A to affect more on node B. On the other hand, user's interactions on node B will not affect much on node A.

Moreover, the force applied on knot O is also calculated as the sum of the two attractive forces created by the two edges.

$$\boxed{F_{O_x} = \frac{(x_A - x_O)}{d_{A,O}} * c_1 * f_{A,O} + \frac{(x_B - x_O)}{d_{O,B}} * c_2 * f_{O,B}}$$ (Formula 3.11)

$$\boxed{F_{O_y} = \frac{(y_A - y_O)}{d_{A,O}} * c_1 * f_{A,O} + \frac{(y_B - y_O)}{d_{O,B}} * c_2 * f_{O,B}}$$ (Formula 3.12)

At each simulation step, the attractive force is calculated for each kind of nodes: normal nodes and knots. As a result of this, all their positions are updated. Knots are internal nodes of curly spring embedded edges, so they are not visible and included in calculating the repulsive force amongst normal nodes.

Furthermore, in order to hide the knot and make our curly edge to look like a real curly one, we make use of Bézier curve (Wikipedia, 2012). According to Weisstein (2012), a Bézier curve is a parametric curve and it is popularly used in computer graphics. In detail, a Bézier curve is defined by "a set of control points P_0 through P_n, where n is called its order (n=1 for linear, 2 for quadratic and 3 for cubic). The first and last control points belong to the curve, but the intermediate control points (if any) generally do not lie on the curve.

Formulas of quadratic Bézier curves and cubic Bézier curves (Wikipedia, 2012)

$$\boxed{B(t) = (1-t)^2 P_0 + 2(1-t)P_1 + t^2 P_2}$$ (Formula 3.13)

$$\boxed{B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3}$$ (Formula 3.14)

In the formula 3.13, a quadratic Bézier curve is the path traced by the function *B (t),* given points P_0, P_1 and P_2 with $t \in [0,1]$ and P_2 is the control point. Simply, the curve starts from P_0 in the direction of P_1, and then bends to arrive at P_2 in the direction from P_1. In other words, the tangents in P_0 and P_2 both pass through P_1 (Figure 3.11).

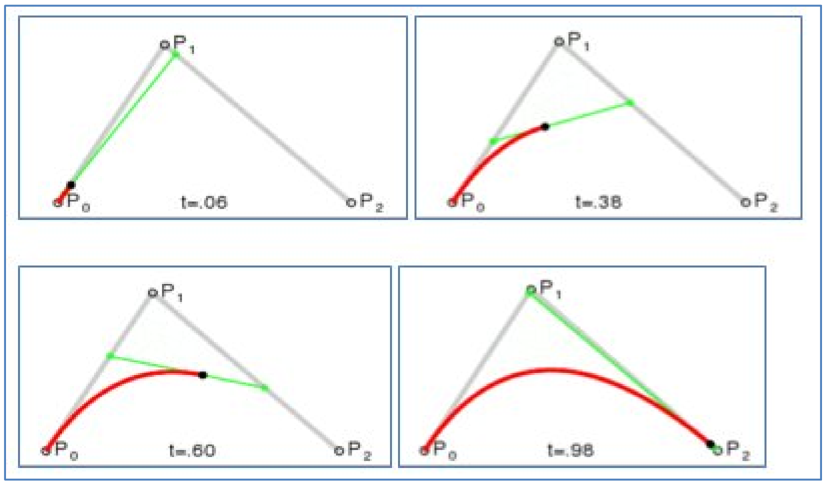


Figure 3.14: Create a quadratic Bézier curve by looping t from 0 to 1 with step =0.01
(Images taken from Wikipedia (2012))

In the formula 3.14, we have a similar formula for a cubic Bézier curve. In this case, P_1 and P_2 are the control points to decide the degree of the curve, so P_0 and P_3 are the only points, which are belonged the curve.

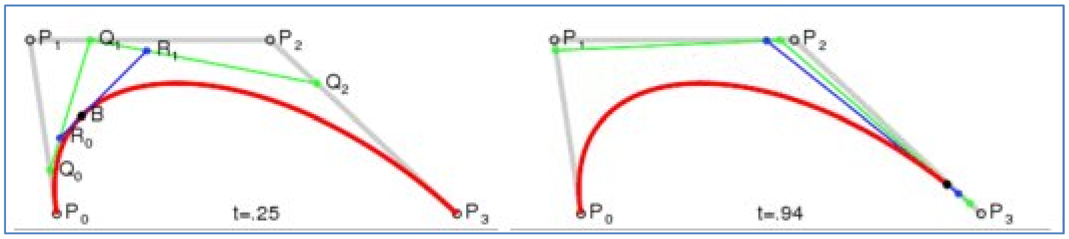


Figure 3.15: Create a cubic Bézier curve by looping t from 0 to 1 with step =0.01
(Images taken from Wikipedia (2012))

We are applying a Bézier curve into our design and it is shown in Figure 3.13.

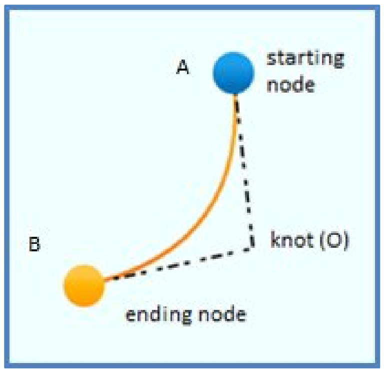


Figure 3.16: Example of applying Bézier curve into our design

From our initial implementation (Figure 3.13), the curve is bent too early, indicating a mediocre approximation. As a result of this, we assign the midpoint of A and O as the starting point for the curve and the midpoint of O and B as the ending point.

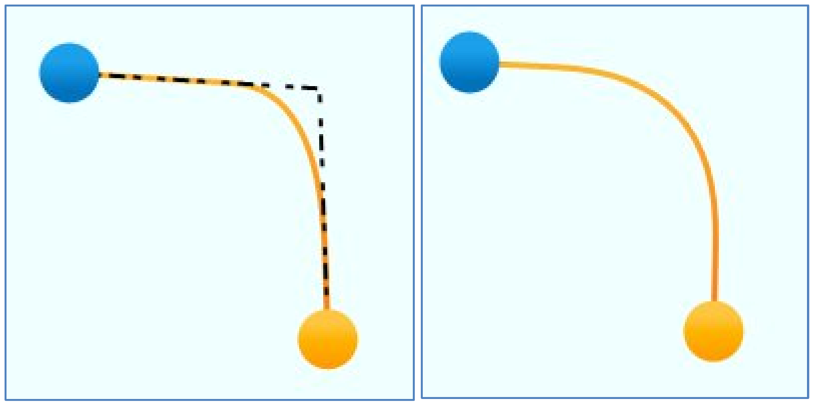


Figure 3.17: An improved version of applying Bézier curve.
A completed version is shown in right image.

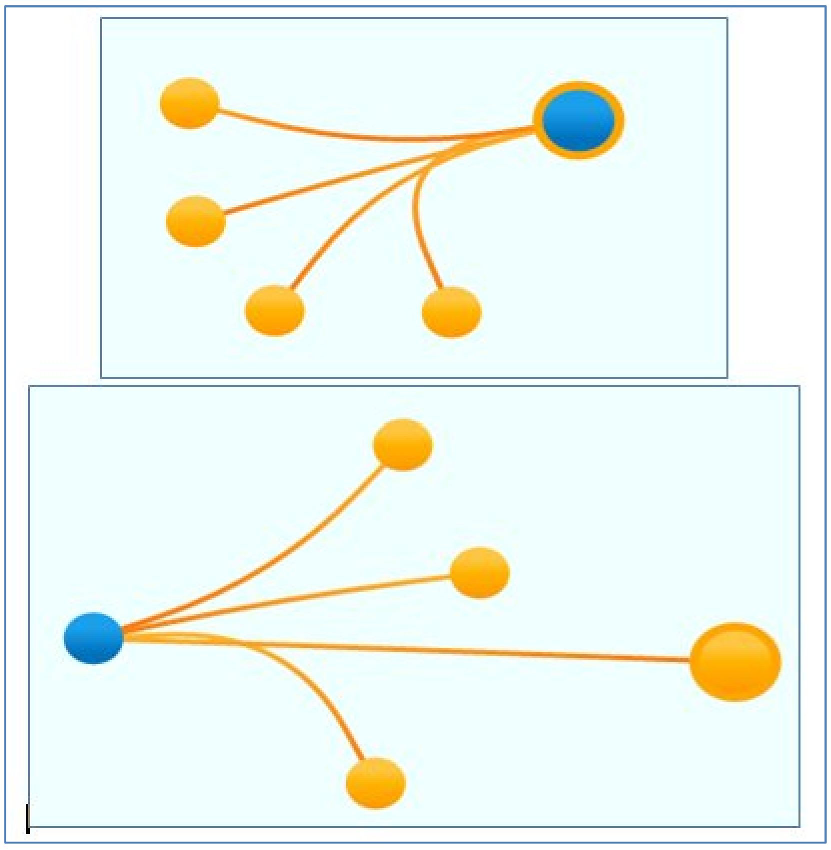


Figure 3.18: Our design in the case that one node has a lot of branches.
By adjusting the attractive force, the root node easily controls its sub nodes (Top image), and its sub-nodes hardly affect their root

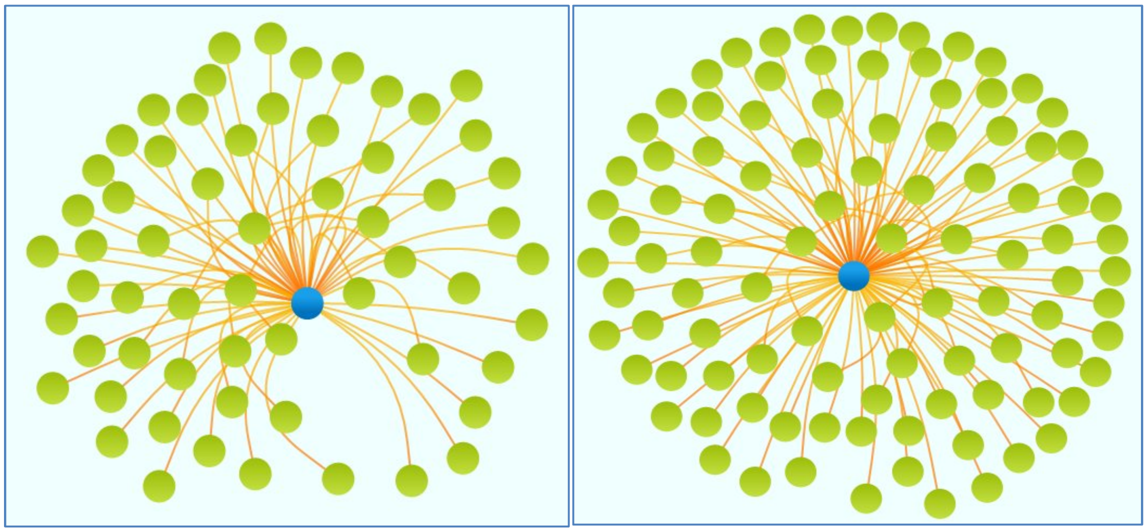


Figure 3.19: Our curly spring embedded edge in the cases that the root branches off 60 sub-nodes and 100 sub nodes.

From Figure 3.16, it is clear that our design works well with the case that one node has a lot of branches (> 60), unlike Eades's and FR algorithms which have a lot of unexpected problems in the case. The key advantage is that because there are joint nodes (knots) between the branching node and its ending node. Changing positions of the ending nodes affect mainly on these knots. As a result of this, they indirectly reduce the amount of attractive force applied on the branching node, which allows it to be interacted smoothly.

In fact, when compared to other people's methods, they solved this branching problem by introducing dummy nodes to group the branches, but this method indeed creates false understanding about the structure of the graph and costs lot of space.

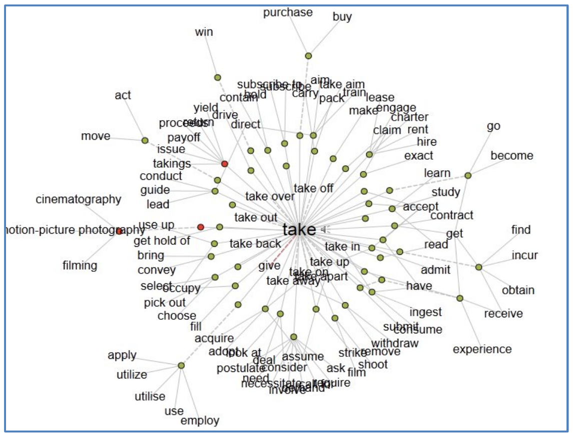


Figure 3.20: An example of solving the branching problem (Image taken from www.visualthesaurus.com/)

Moreover, our curly edge allows designers to control its length to encode information. In fact, the curly edge is more elegant than the straight one in some visualization applications (Figure 3.21).

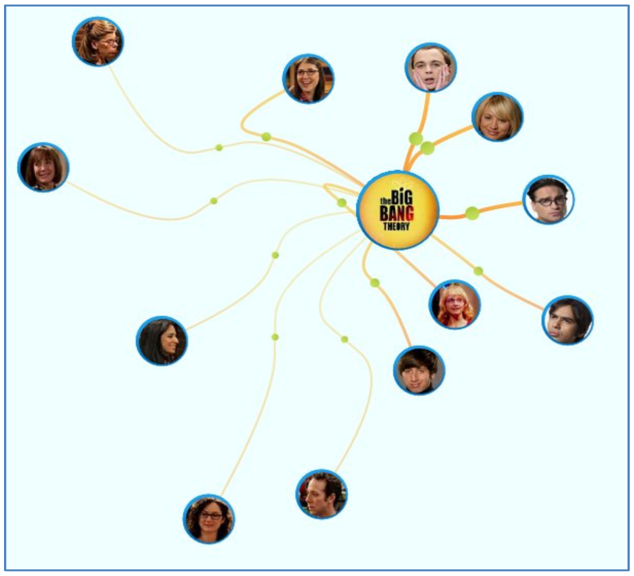


Figure 3.21: Visualize characters in The Big Bang Theory Show.
A lot of information is encoded in the curly edge, the shorter the length and the larger the thickness are, the more frequently the character appears in the show. So, it's easy to see who main characters are.

In fact, our curly edge works well with the traditional straight edge. Particularly, it complements the straight edge. In detail, it is very useful in the case in which graph nodes are belonged to multi-categories. To the best of our knowledge, people solve this problem by using multi-colors on these nodes in order to denote different categories (Itoh, Muelder, Ma and Sese, 2009).

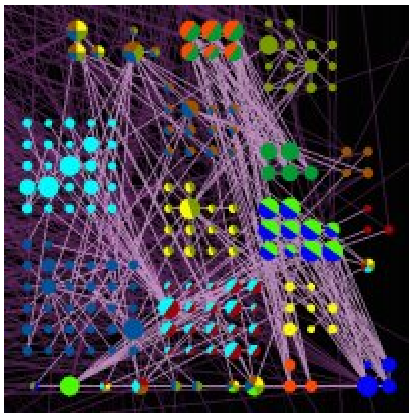


Figure 3.22: An example of a graph in which nodes are belonged to multi-categories.
(Image is taken from Itoh, Muelder, Ma and Sese (2009)).

By using our curly edges, designers can make connections between multi-categoried nodes with their minor categories. It is clear to see that our curly edges make such graphs more understandable and still maintain the symmetry which is created by the straight edges.

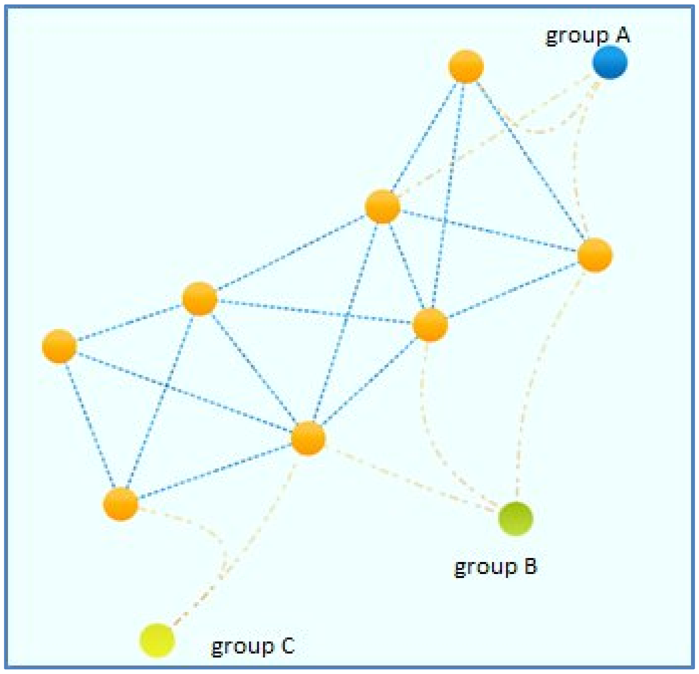


Figure 3.23: An example of using our curly edge to visualize nodes belonged to multi categories.

Moreover, the curly style is also suitable to use in flow visualizations, which is proved by work of Doantam, Xiao, Yeh et al. (2005) on flow maps in which curly edges are used to create smooth flows and allow placing nodes in correct geographical locations.

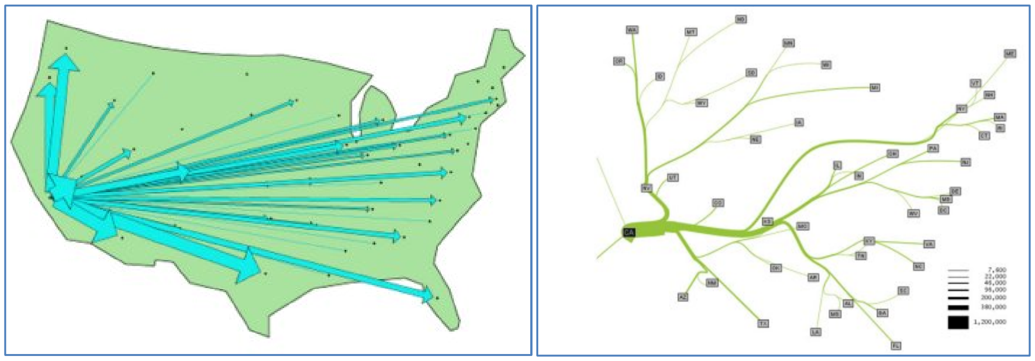


Figure 3.24: An example of curly edges used in a flow map
(Image taken from Doantam, Xiao, Yeh et al. (2005))

In our application, the curly edge can also be used to visualize a flow of the number of frequencies from a root to its sub-nodes which display results. As we mentioned, the popular of a lexical phrase is indicated by its frequency. For example, the frequency of phrase “cause trouble” of the lexical pattern “cause +Noun” is 250000, which is big, so this phrase is used commonly. From the collocational database, we can recursively calculates the frequency of each lexical pattern and then each category (used as noun or used as verb), finally each word (take, or cause...) and store this information back in the database. As a result of this, this information can be used to construct a flow from a root to its sub nodes by using the thickness of our curly edges.

For example, Figure 3.25 show a flow from a root node of the input word “live” to its sub nodes which display categories. It is clear that users can recognize quickly which category the input word “live” is used more commonly. Logically, they may expect to more lexical forms of “Used as Verb” category than the ones of “Used as Adjective” category. This information is indeed useful because it helps users to improve decision making in graph explorations and learning process.

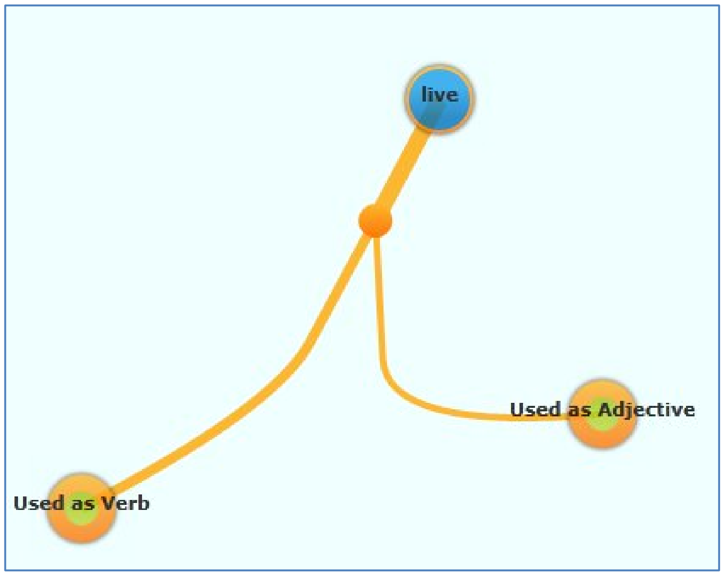


Figure 3.25: Visual a flow of frequencies from a root node to its sub-nodes

Figure 3.26 below shows further expansion of the navigating nodes. It can be interpreted that some particular lexical combinations are being used more others.

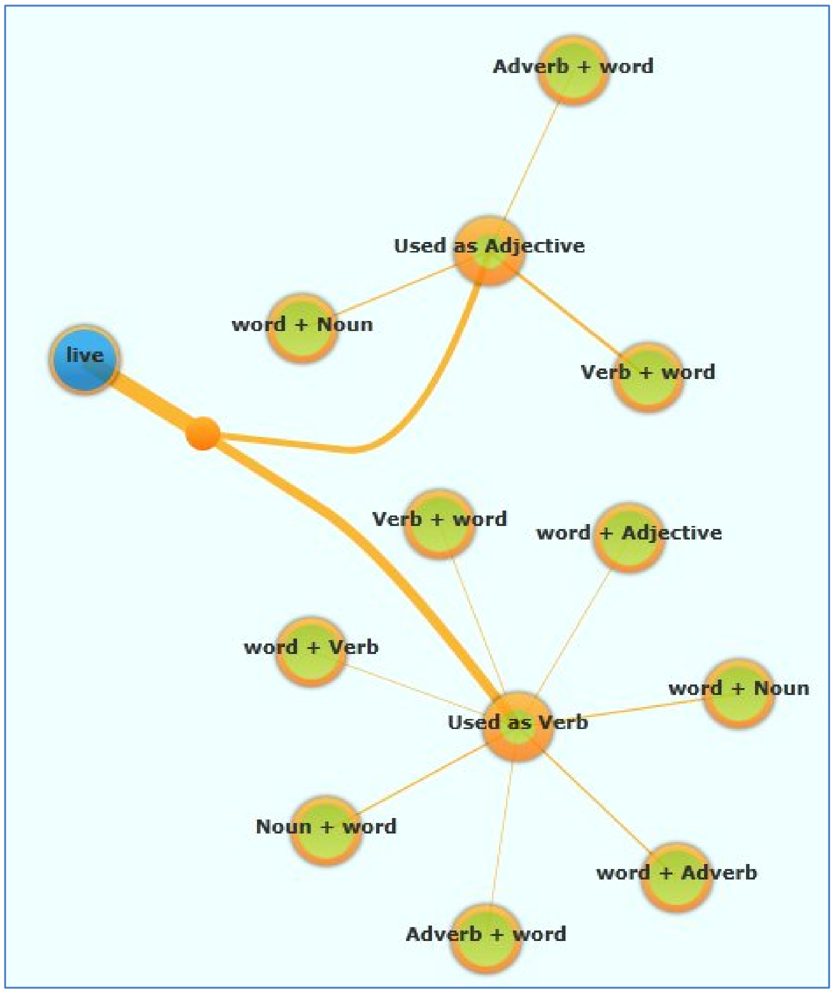


Figure 3.26: Further expansion of the navigating nodes with flows visualized

To sum up, our curly spring embedded edge indeed solves the branching problem and allows designers to utilize its properties such as length and thickness to encode information. By using its internal node to receive the attractive force, it reduces dramatically amount of the attractive force applied on the branching node, making users' interactions on these nodes more smoothly. Particularly, it complements the current straight spring embedded edge in order to visualize more specific information on force-directed graphs such as flow visualization and or multi-categorized nodes.

With the new curly edge, our algorithm to construct the graph layout can be summarized as follows:

Algorithm 4: Our force-directed algorithm

Input: a set of vertices V and set of edges E - G = (V, E)
Output: a nice drawing without zero angular resolution.
Require: A 2d-quadtree and *tempForce[|V|]* stores temporary forces of all vertices; *newPos* and *oldPos* record the new and old positions of all vertices, respectively.

```
1.  assign initial locations of vertices in V
2.  initial the quadtree and place all newPos in
3.  while converged ≠ 1 or the maximum iteration number is achieved do
4.     converged ← 1
5.     oldPos ← newPos
6.     initialize tempForce[|V|] as zeros matrix

      //calculate repulsive force between edges from the same starting vertices
7.     for each vertex v in V do
8.        if the degree of vertex v is at least two then
9.           for each pair ($e_i$, $e_j$) do
                // where edge $e_i$ = (v, vi) and edge ej = (v, $v_j$) are neighboring edges incident to v
                //calculate the repulsive force f according to formula 3.4
10.             tempForce[$v_i$] ← tempForce[$v_i$] + f
11.             tempForce[$v_j$] ← tempForce[$v_j$] − f
12.          end for
13.       end if
14.    end for
       //calculate the attractive force applied on two connected vertices
15.    for each edge e = ($v_i$,$v_j$) in E do
          //calculate the spring force of edge e according to formula 3.2 and 3.3
16.       tempForce[$v_i$] ← tempForce[$v_i$] + $f_{a1}$    // $f_{a1}$ is for starting vertex
17.       tempForce[$v_j$] ← tempForce[$v_j$] + $f_{a2}$    // $f_{a2}$ is for ending vertex
18.    end for
       //calculate the repulsive force between vertices
19.    for each vertex v in V do
20.      query from the quadtree a list of vertices Ve which are close enough from (distance < r)
21.       for each $v_i$ in Ve do
22.             tempForce[v] ← tempForce[v] + $f_r$(v,$v_i$)

       //update newPos with tempForce
23.    for each vertex v in V do
         // $c_6$ (t) is a constant to control the magnitude of movement
24.      newPos ← newPos + min(tempForce[v],$c_6$ (t))
25.    end for
26.    if (||newPos – oldPos|| > ε then
27.       converged ←0
28.    end if
29. end while
```

Algorithm 4: Our force-directed algorithm

3.2.4. Design and implement interface

a) Design of the graph layout

As we mentioned above, after users input their searched word, we use a graph layout to guide users to explore all lexical combinations of the input word. In addition, all navigating nodes are animated in chronological order to help users to increase engagement level and keep them oriented; intuitively, it makes users to sense that input information has been processed.

Furthermore, we use our curly edge to show top 30 resulted phrases of each lexical form. In order to avoid creating visual clutter and keep our graph more readable. We are not labeling sub-nodes which represent the resulted phrases and scale them down, but we indeed distinguish high-frequent phrases by the opacity, length and thickness of the edges that they connect. In detail, high-frequent phrases have their edges long; moreover, their edges also have high opacity and large thickness.

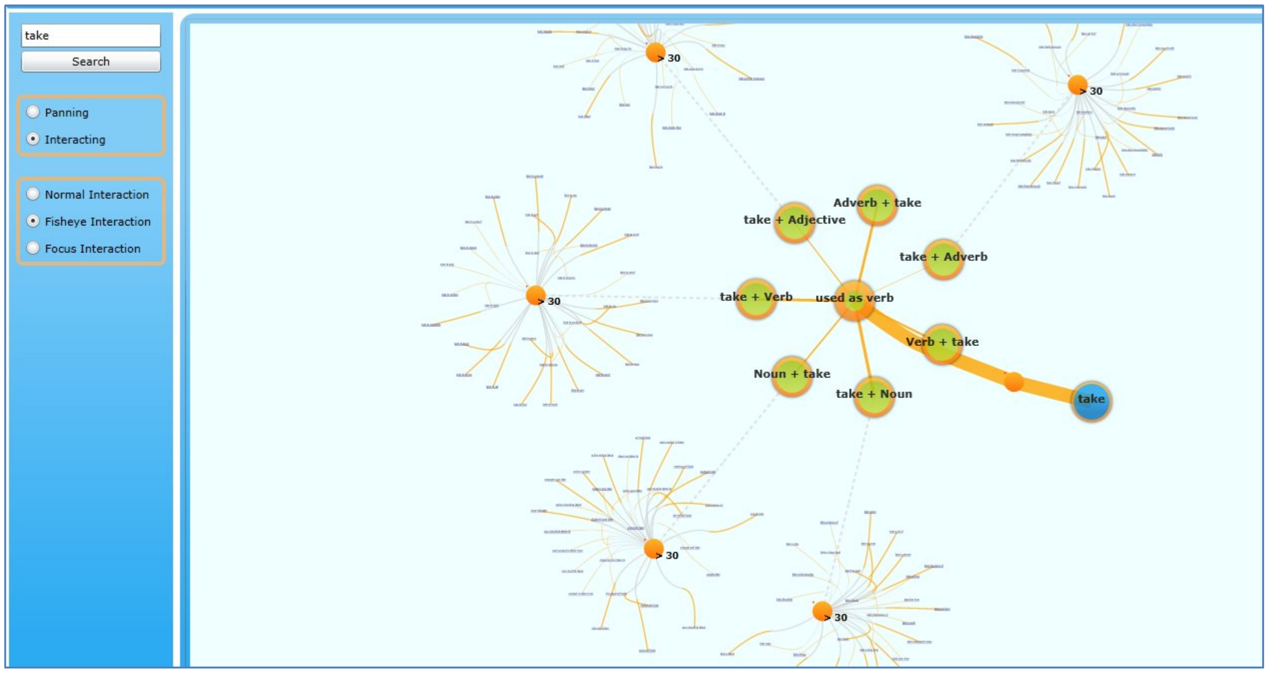


Figure 3.27: Our graph interface
(The numbers shown on orange nodes indicate the number of resulted phrases)

Figure 3.27 shows our graph implementation. Its characteristics allow us to see “the whole picture” of the input; particularly, they help users to easily compare between lexical combinations. Furthermore, our implementation is able to display more than two words' results (Figure 3.25).

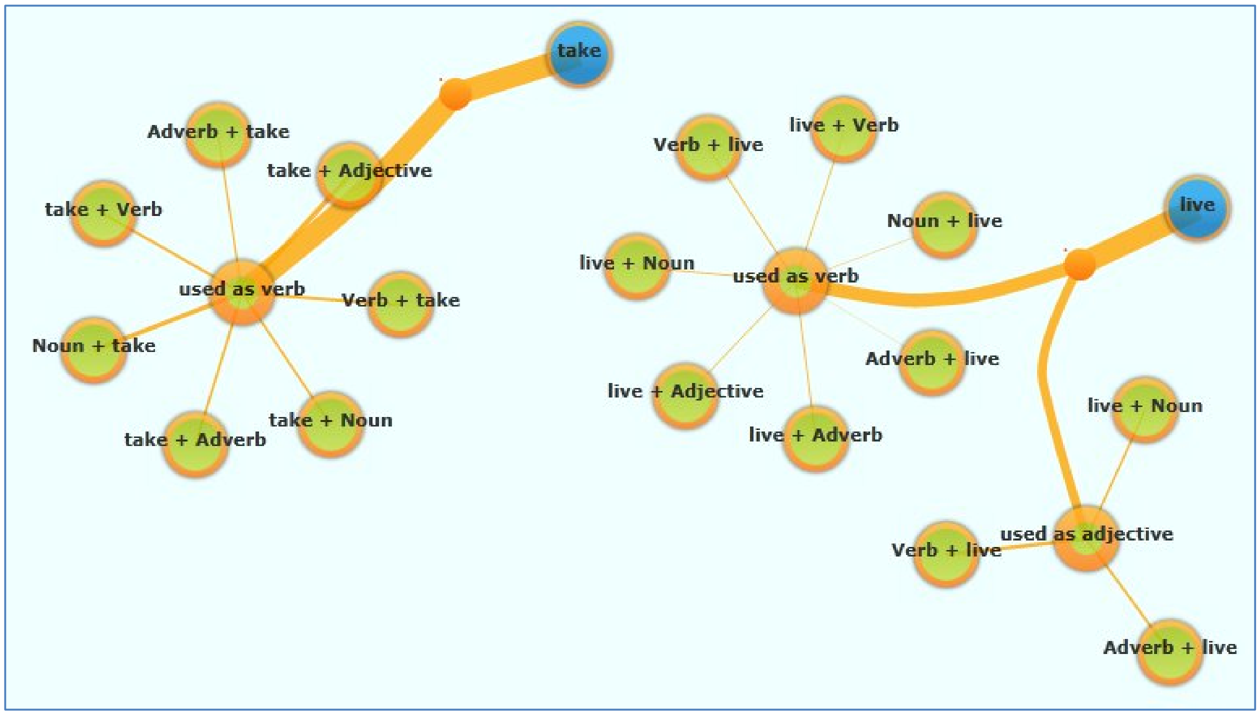


Figure 3.28: Display two words on the same layout

b) <u>Design interactive techniques</u>

Furthermore, in order to support users to interact with the graph system, we implement panning technique to allow the users to change visual area (viewport) quickly. In addition, we also implement three kinds of interactions which are: normal interaction, focus interaction and fisheye interaction.

In detail, normal interaction will circle and scale the selected phrase up to a good visual size which allows users to see properly. Focus interaction is similar to normal one, but it highlights nodes and edges that connect to the selected node and reduce opacity of other edges and nodes. Fisheye technique; on the other hand, not only scale up the selected node but other nodes which belong to the fisheye's area. In order to make it more like a fisheye lens, the scale values of the other related nodes are determined by the formula 3.15

Formula 3.15 to determine scale values of the nodes that belong to the fisheye area.

$$\boxed{s(d) = s_0 * (1 - d / r) + c_0} \qquad \text{(Formula 3.15)}$$

In formla 3.15, s_0 is the scale of the selected node (center), d is the distance between the calculated node and center node, r is the fisheye' radius and c_0 is the scale for nodes at unselected state. In fact, this formula is a simple version of the one that Sarkar and Brown (1992) introduced.

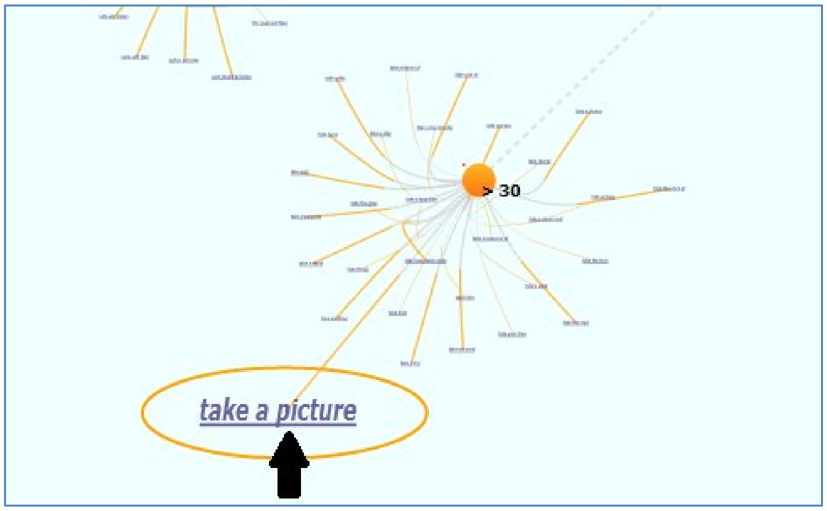


Figure 3.29: Normal interaction.
We circle the selected phrase

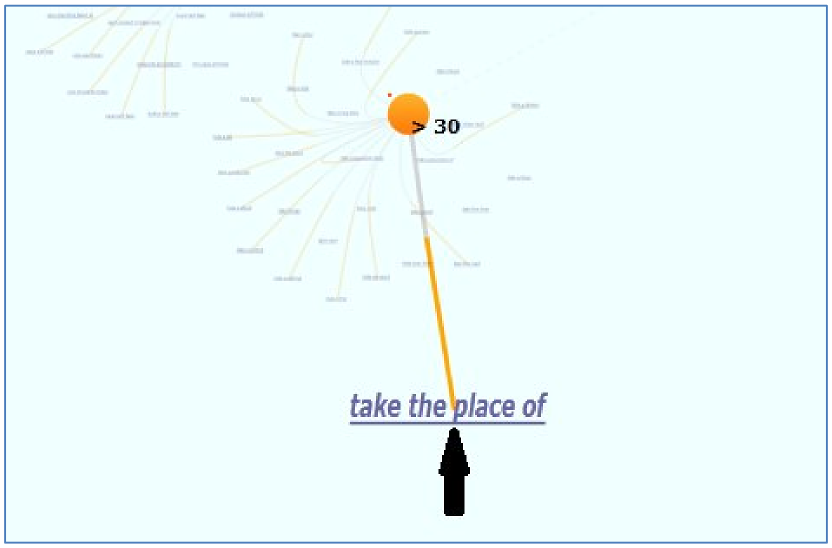


Figure 3.30: Focus interaction
We highlight selected node and its edge and reduce the opacity of other nodes and edges

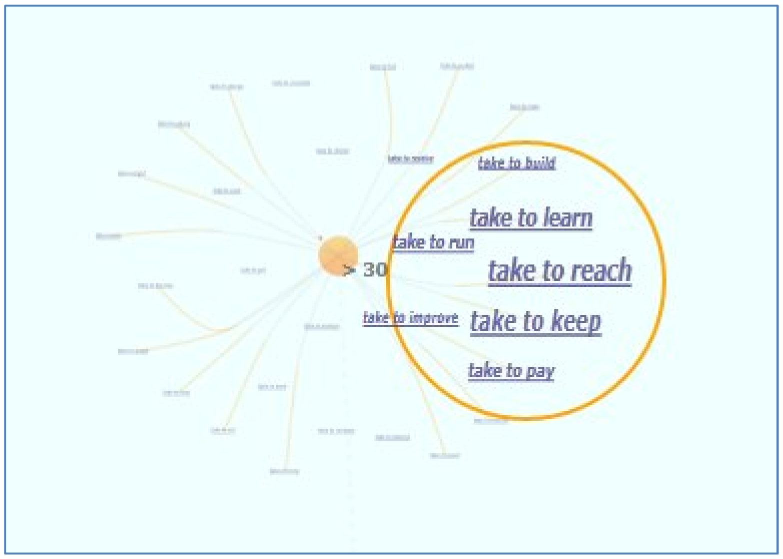
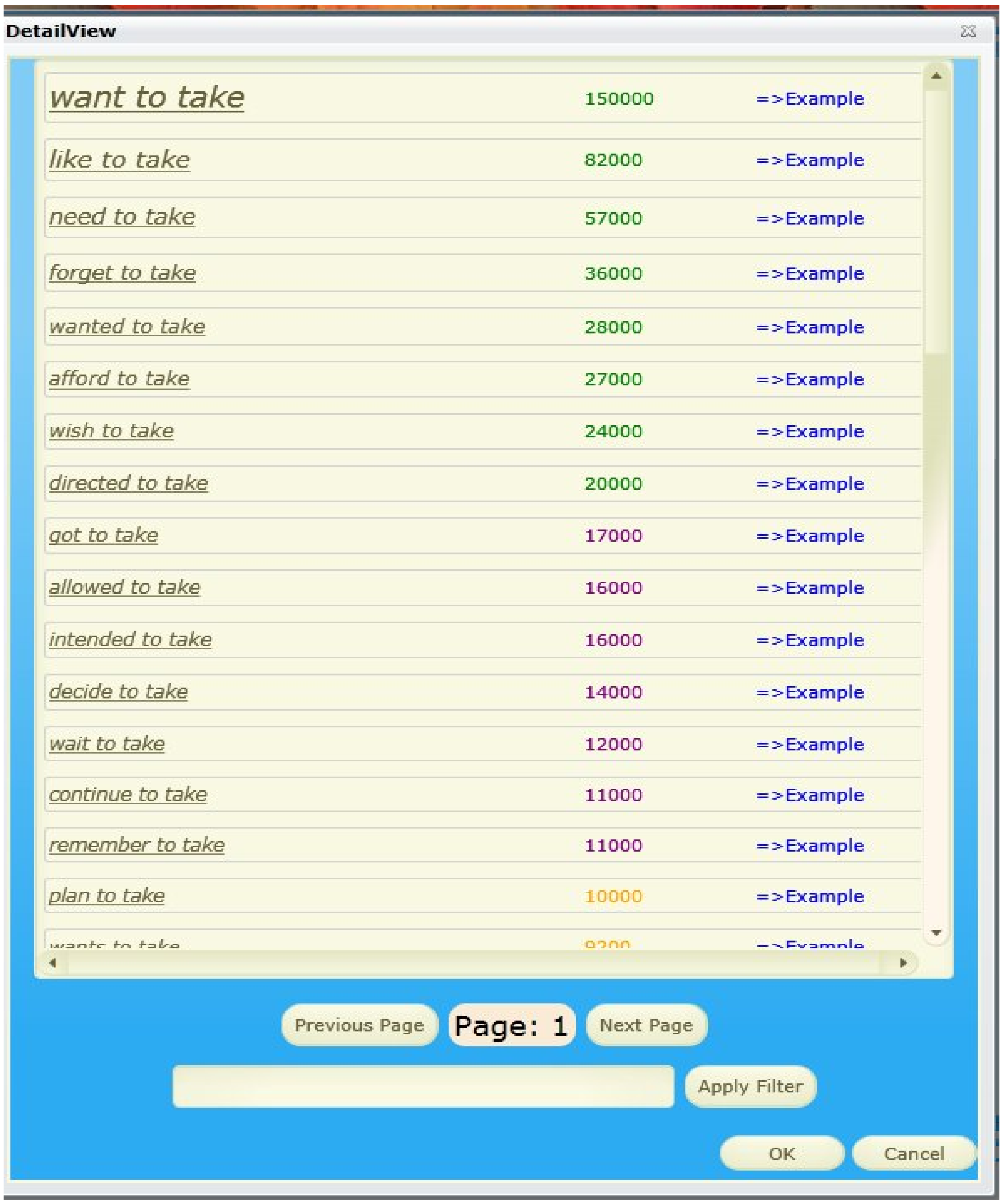


Figure 3.31: Fisheye interaction on nodes labeling resulted phrases

c) <u>Design a detail view</u>

The graph layout provides good navigation for the users and allows them to see a whole picture. We limit the number of nodes that label resulted phrases to 30 in order to keep graph readable. In order to show resulted phrases (> 30th phrase), we also implement a detail view for complete showing. If the users double-click on an orange branching node (Figure 3.29), it will pop up a detail to show all resulted phrases.



want to take	150000	=>Example
like to take	82000	=>Example
need to take	57000	=>Example
forget to take	36000	=>Example
wanted to take	28000	=>Example
afford to take	27000	=>Example
wish to take	24000	=>Example
directed to take	20000	=>Example
got to take	17000	=>Example
allowed to take	16000	=>Example
intended to take	16000	=>Example
decide to take	14000	=>Example
wait to take	12000	=>Example
continue to take	11000	=>Example
remember to take	11000	=>Example
plan to take	10000	=>Example
wants to take	9200	=>Example

Figure 3.32: A detail view of lexical pattern "Verb +take"

From Figure 3.32, it is clear that we encode the difference of the frequency in font size of the resulted phrases. High-frequent resulted phrases should have large font in order for users to recognize easily. In detail, we follow work of Knautz, Soubusta, and Stock (2010) on web navigating tags. Even although they haven't clarified that why tag clouds are popular, their experiments indeed show that large tags are more recognizable for users. As a result of that, we use the frequency as input variable to produce suitable font size for the resulted phrases (Formula 3.16). With this method, popular phrases are easily recognized by the users. So, it will speed up and improve their learning.

$$\boxed{s = s_0(1 + Max(f / f_{\max}, \Delta_0)} \quad \text{(Formula 3.16)}$$

In formula 3.16, s is the font size of input phrase; f is the phrase's frequency. In addition, s_0 is the base font size, $f_{\max}$ is the maximum frequency and Δ_0 is the maximum amount allowed to add to s_0.

Technically, we understand that our application is running on-line. So, we use paging technique to fetch resulted data only when users require (by clicking paging buttons). Therefore, we reduce dramatically the data retrieving time. Moreover, we also implement a filtering function which allows flexibly data manipulation

3.2.5. Technical evaluation of our implementation

For this application, we mainly use Silverlight technology. This is a web tool from Microsoft to create rich client web applications. It is similar to Flash which allows designers to create rich animation applications.

In order to visualize completely all aspects of information, we heavily use quite a number of advanced graphic objects such as gradient colors, graphic paths, opacity property and Bézier curves. Technically, our current implementation focuses on delivering good visual appearance. But our implementation is running online at client side so memory and retrieving data ability are limited. As a result of this, the number of object that we can draw is limited. In order to solve that problem, we implement a number of practical techniques to improve our performance. Firstly, we improve the calculation by using a quadtree that we modified in section 3.2.2. Secondly, we use "virtual canvas" technique — a technique allows designers to draw large number of object on a canvas.

In detail, if all the nodes cover an area (0, 0, 5000, 5000) and the viewport which the user focuses on covers the area (2000, 2000, 2500, 2500); because all nodes are placed in a quadtree, we can do a query in the quadtree to list all nodes belong to the area (2000 – offset, 2000-offset, 2500 + offset, 2500 + offset) with offset = 400. Then, we only draw on the canvas these nodes and their associated edges. When the user changes the viewport by panning, we remove these old nodes and edges from the canvas and perform a new query with new area. However, if there are more than 480 nodes locating in the viewport, our performance will drop dramatically; particularly, low response and animation glitches are likely to occur when users interact.

In fact, technical features on our implementation can be summarized as follows:

Working state	Number of nodes afforded to create	Interactive state
Normal state (working nicely)	Up to 300	Three kinds of interactions are working normally. They response fast to users' interactions.
Maximum state	500	- Normal interaction is still working normally. - Focus interaction responses slowly. - Fisheye interaction responses slowest and scaling function is not performing smoothly.

Table 3.1: Summary of our graph implementation

In fact, we also did some experiments to improve drawing capacity by reducing visual appearance; however, it also at the same time reduces quite a number of means in which we can encode information.

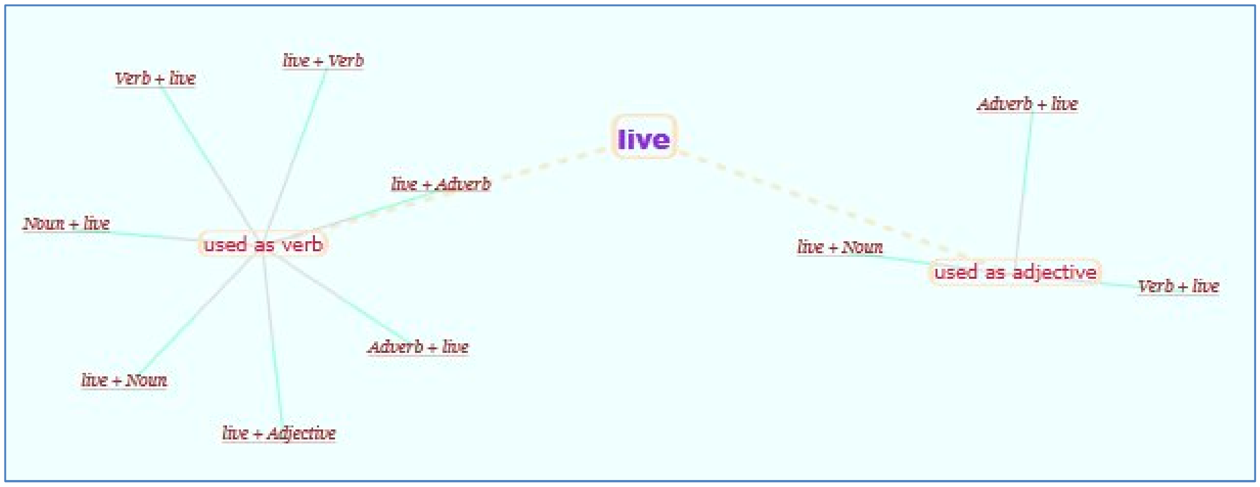


Figure 3.33: A simple implementation

From Figure 3.33, it is easy to see that in new implementation, flow visualization, gradient colors and opacity attribute are removed. As a result of this, the quality of visual appearance is not high, but it can handle drawing large number of objects. From our experiment, the number of nodes it can draw up to 2000.

To sum up, due to being an online application, we have to make compromise between visual appearance and performance. All though, our implementation has a limit of the number of object displayed, it indeed allows a lot of information attributes visualized to users in order to improve their learning and engagement.

3.2.6. Usability evaluation of our implementation

In order to evaluate the implementation, we do some usability tests on a group of 6 people. In detail, these participants are students and not English native speakers. Two participants are at advanced level and the others are at immediate level. We do not include beginners in our study, because from our research beginners usually do not pay attention to collocational usage.

First of all, we inform all participants that how collocational knowledge is important and useful to second-language learners, and then introduce them shortly the interface. We let these participants interact with the interface, observe their interactions and take note of their opinions. The result of the experiment can be summarized as follows:

Number of Participants for each type of opinion	Opinions	English level of participants
2	Enjoy the interface (feel comfortable)	1: Advanced and 1: immediate
3	Feel all right with the interface	2: Immediate and 1: Advanced
1	Feel unfamiliar, prefer an interface like the one in dictionary software	1: Immediate

Table 3.2: Table of participants' opinions.

From Table 3.2, our graph implementation can be seen as easy for interaction. Moreover, we also receive some interesting feedbacks from participants. For example, when they select a resulted phrase, they expect to see a tooltip to explain how the phrase means. Moreover, the advanced users expect to see some relationships between input words when our application displays them on the same layout. They also expect to open more than one detail views at the same time. These feedbacks are indeed very useful for future development.

Furthermore, we also let the participant interact with the original application designed by Wu for comparison. Our implementation is more favorable by most participants due to our good navigation. However, there are some features from original application that participants prefer more; for example, they want our detail view to show more results as the original application does.

Overall, even though the number of participants is small, the feedbacks we get from them are valuable. We believe that our current implementation meets users' demands. Moreover, with these useful feedbacks, we can develop further our application to satisfy users' expectations.

Chapter 4 –Conclusion

4.1.Conclusion

In this thesis, we represent our approach to visualize online-collocational dictionary by utilizing graph representation. We consider using a force-directed model due to its simplicity and flexibility in interaction. However, after being studied and experimented comprehensively, all current force-directed algorithms have their own advantages and drawbacks. Particularly, these drawbacks make them not suitable to be applied directly to our visualization application. As a result of this, we develop our graph component (curly spring embedded edge), implement some modifications on some existing algorithms, and then combine them in order to make an algorithm that is more suitable to our purposes. Furthermore, some practical techniques are also applied in order to produce a good visualization application in which all important aspects of information are represented noticeably.

Benefits of using graph representation has been proved by a lot of researches, and in our application, it allows us to display a good navigating system which indeed facilitate graph exploration and language learning for users. After being tested by a small group of participants, our application can be considered to meet our expectation and satisfy most users' demands. With their valuable feedbacks, we can develop further application in order to make it more convenient to all users.

4.2.Future Work

As we mentioned before, current force-directed models are suitable to visualize general graphs, but they do not control well the output of graph components such as edges. As a result of this, they do not utilize well these components' attributes in order to visualize fully information. Our force-directed model, on the other hand, fits specifically well to unconnected graphs and allows designers to use edge's length to visualize different levels of graph nodes' relationships.

Moreover, our curly spring embedded edges not only solve the branching problem, but also complement the current straight spring embedded edge to visualize fully some particular graphs such as graphs with multi-categorized nodes or graphs with flow visualization. As a result of this, our future research will study what kind of graph visualization applications in which our curly edge will be utilized all its potential. In fact, we are considering that our curly edge can be applied in edge-plucking technique which is proposed by Wong and Carpendale (2005) and force-directed edge bundling which is introduced by Holten and van Wijk (2005), if we improve the curly edge by allowing internal nodes to able to repel or attract each other.

Furthermore, our implementation also has a lot opening way that we can develop and improve. In fact, all feedbacks from participants are indeed valuable, and they help us to develop our application in a proper way in order to makes it more suitable for users' learning and exploration.

Chapter 5 – Bibliography

1. Barnes, J. & Hut, P. (1986). A hierarchical O (NlogN) force-calculation algorithm. *Nature, 324*, 446-449.

2. Battista, G. D., Eades, P., Tamassia, R., & Tollis, G.I. (1999). Algorithms for the visualization of graphs. Princeton, NJ. Prentice-Hall.

3. Baudish, P., and Boltman, A. (1999). Does animation help users build mental maps of spatial information? Proceedings of 1999 IEEE Information Visualization, San Francisco, CA, 28-35

4. Benson, M., Benson, E., and Ilsen, R.F. (1986). The BBI combinatory dictionary of English: A guide to word combinations. Amsterdam/Philadelphia: John Benjamins.

5. Chan, A.Y.W. (2004). Syntactic transfer: Evidence from the interlanguage of Hong Kong Chinese ESL learners. *The Modern Language Journal*, *88*(1), 56-74.

6. Chen, C. (2004). Information Visualization: Beyond the Horizon. Singapore: Springer.

7. Cockburn, A., Karlson, A., & Bederson, B.B. (2008). A review of overview + detail, zooming, and focus + context interfaces. *Journal ACM Computing Surveys, 41*(1). doi: 10.1145/1456650.1456652

8. Darvishi, S. (2011). The investigation of collocational errors in university students' writing majoring in English. *International Conference on Education, Research and Innovation, 18*(12), 52-57

9. Doantam, P., Xiao, L., Yeh, R., Hanrahan, P., and Winograd, T. (2005). Flow map layout. Proceedings of IEEE 2005 Symposium Information Visualization, 219-224

10. Eades, P. (1984). A heuristic for graph drawing. *Congressus Nutnerantiunt, 42*, 149-160.

11. Eades, P., Huang, L.M., Wang, J., & China, P.R. (1998). Online animated graph drawing using a modified spring algorithm. *Journal of Visual Languages and Computing, 9*, 17-28

12. Gonzales, C. (1996). Does animation in user interfaces improve decision making? Proceedings of ACM CHI 1996, Vancouver, BC, 27-34

13. Frishman, Y. & Tal, A. (2008). Online dynamic graph drawing. *IEEE Transaction on Visualization and Computer Graphics, 14*(4), 727-740.

14. Fruchterman, M.J.T., & Reingold, M.E. (1991). Graph drawing by force-directed placement. *Software Practice and Experience Journal, 21*(11), 1129-1164

15. Hadany, R. & Harel, D. (2001). A multi-scale algorithm for drawing graphs nicely. *Discrete Applied Mathematics, 113*, 3-21.

16. Hadany, R. & Harel, D. (2002). A fast multi-scale method for drawing large graphs. *J. Graph Algorithms and Applications, 6*, 179-202.

17. Heer, J., Boyd, D. (2005). Vizster: Visualizing online social networks. Proceedings of the 2005 IEEE Symposium on Information Visualization, Minneapolis, MN, 32-39

18. Heer, J., & Roberton, G. (2007). Animated transitions in statistical data graphics. *Journal IEEE Transactions on Visualization and Computer Graphics, 13*(6). doi: 10.1109/TVCG.2007.70539

19. Hendrickson, B. & Leland, R. A multilevel algorithm for partitioning graphs. Proceeding of Supercomputing '95, New York, NY. doi: 10.1145/224170.224228

20. Herman, I., Melancon, G., Marshall, M.S. (2000). Graph visualization and navigation in information visualization: a survey. *IEEE Transactions on Visualization and Computer Graphics, 6*(1), 24-43.

21. Holten, D. (2006). Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics (Proc. of INFOVIS'06), 12*(5), 741–748.

22. Holten, D., and van Wijk, J.J. (2009). Force-directed edge bundling for graph visualization. Proceedings of the 11th Eurographics/IEEE-VGTC Symposium on Visualization, 28(3), 983-990.

23. Holten, D. and van Wijk, J.J. (2009). A user study on visualizing directed edges in graphs. Proceedings of the SIGCHI Conference on Human Factor in Computing Systems (CHI'09), 2299-2308

24. Hu, F.Y. (2005). Efficient and high quality force-directed graph drawing. *The Mathematica Journal, 10*, 37-71

25. Hu, F.Y. (2011). Algorithms for visualizing large networks. *Combinatorial Scientific Computing*. Retrieved from http:// www2.research.att.com/~yifanhu/PUB/ch16.pdf

26. Huang, W., Eades, P., Hong, S.H and Lin, C.C. (2010). Improving Force-Directed Graph Drawings by Making Compromises between Aesthetics. Proceedings of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'10). IEEE Computer Society, Washington, DC, USA, 176-183.

27. Interactive language learning – flax library. Collocation Activities. Retrieved from http://flax2.nzdl.org/greenstone3/flax?a=p&sa=about&c=colloact

28. Itoh, T., Muelder, C., Ma, K., and Sese, J. (2009). A hybrid space-filling and force-directed layout method for visualizing multiple-category graphs. Proceedings of the 2009 IEEE Pacific Visualization Symposium, Beijing, 121-128

29. Jiang, N. (2004). Semantic transfer and its implications for vocabulary teaching in a second language. *The Modern Language Journal 88* (3): 416–432.

30. Johannes, F. (2005). *Interactive Visualization of Large Graphs*. (Doctoral dissertation). Retrieved from http:// thevcl.com/papers/thesis.pdf

31. Kamada, T. & Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information Processing Letters, 31*(1), 7-15.

32. Karypis, G., and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing, 20*, 359-392.

33. Kim, K., Ko, S., Elmqvist, N., & Ebert, S.D. (2011). WordBridge: Using composite tag clouds in node-link diagrams for visualizing content and relations in text corpora. Proceedings of the 44th Hawaii International Conference on System Sciences, 1-8

34. Knautz, K., Soubusta, S., and Stock, G.W. (2010). Tag Clusters as Information Retrieval Interfaces. Proceedings of the 2010 43rd Hawaii International Conference on System Sciences, Honolulu, HI, 1-10

35. Kobourov, G. S. (2004). *Force-directed drawing algorithms*. Retrieved from http://www.cs.brown.edu/~rt/gdhandbook/chapters/force-directed.pdf

36. Lewis, M. (1997). Implementing the Lexical approach. Hove: Language Teaching Publications.

37. Liang, J. & Huang, L.M. (2010). Highlighting in Information Visualization: a Survey. Proceedings of the 14th International Conference Information Visualization, London, 79-85

38. Li, C. (2005). A study of collocational error types in ESL/EFL college learners' writing. Retrieved from http://ethesys.lib.mcu.edu.tw/ETD-db/ETD-search/view_etd?URN=etd-0730105-205237

39. Lin, C., & Yen, H. (2011). A new force-directed graph drawing method based on edge-edge repulsion. *Journal of Visual Languages and Computing*, *29*(1), 29-42.

40. Liu, C.P., (1999). An analysis of collocational errors in EFL writings. Proceedings of the 8th International Symposium on English Teaching, 483-494

41. Nakata, T. (2006). English collocation learning through meaning-focused and form-focused activities: Interactions of activity types and L1-L2 congruence. Proceedings of the 11th Conference of Pan-Pacific Association of Applied Linguistics, 154-168

42. Plafzner, S. & Gibbon, P. (1996). Many-Body Tree Methods in Physics. Cambridge University Press.

43. Pulo, K.J. (2001). Recursive space decompositions in force-directed graph drawing algorithms. *Conferences in Research and Practice in Information Technology Series, 9*, 95-102.

44. Quigley, A. & Eades, P. (2000). Fade: Graph drawing, clustering, and visual abstraction. *Lecture Notes in Computer Science, 1984*, 183-196.

45. Sarkar, M., & Brown, M.H. (1992). Graphical fisheye views of graphs. Proceedings of Human Factors in Computing Systems Conference, 83-91

46. Steele, J., & Iliinsky, N. (2010). Beautiful Visualization: Looking at Data through the Eyes of Experts. Sebastopol, CA: O'Reilly Media.

47. Stuart, K.C., Jock, D.M., and Shneiderman, B. (1999). Readings in Information Visualization: Using Vision to Think. San Francisco, CA: Morgan Kaufmann.

48. Tunkelang, D. (1999). *A Numerical Optimization Approach to General Graph Drawing*. (Doctoral dissertation). Retrieved from http://reportsarchive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-189.pdf

49. Tversky, B., Morrison, J., and Betrancourt, M. (2002). Animation: Can it facilitate? *Int. J. Human-Computer Studies, 57*(4), 247-262

50. Viegas, B.F., Wattenberg, M., and Feinberg, J. (2009). Participatory Visualization with Wordle. *Journal IEEE Transactions on Visualization and Computer Graphics, 15*(6), 1137-1144

51. Walshaw, C. (2003). A multilevel algorithm for force-directed graph drawing. *J. Graph Algorithms and Applications, 7*(3), 253-285.

52. Ware, C. (2008). *Visual Thinking: for Design*. Massachusetts: Morgan Kaufmann.

53. Ware, C. (2004). *Information Visualization: Perception for Design* (2nd ed.). San Francisco, CA: Morgan Kaufmann.

54. Wikipedia. (2012). *Bézier curve*. Retrieved from http://en.wikipedia.org/wiki/Bézier_curve

55. Weisstein, E.W. (2012). *Bézier curve*. Retrieved from MathWord—A Wolfram web resource http://mathworld.wolfram.com/BezierCurve.html

56. Wong N., Carpendale, S., & Greenberg S. (2003). Edge-Lens: An Interactive Method for Managing Edge Congestion in Graphs. Proceedings of the 2003 IEEE Symposium on Information Visualization, 51–58.

57. Wong N., Carpendale, S. (2005). Supporting interactive graph exploration with edge plucking. Proceedings of the 2005 IEEE Symposium on Information Visualization, 51–52.

58. Wu, S., Witten, H.I., and Franken, M. (2010). Utilizing lexical data from a Web-derived corpus to expand productive collocation knowledge. *ReCALL, 22*(1), 83-102

59. Wu, W.S. (1996). Lexical collocations: One way to make passive vocabulary active. Paper presented at The Eleventh Conference on English Teaching and Learning in the Republic of China, Taipei, 61-480

60. Yuan, G., Dancheng, L., Chunyan, H., and Zhiliang, Z. (2009). An improved network topology auto-layout solution based on force-directed placement. Proceedings of the 2009 Ninth International Conference on Hybrid Intelligent Systems, Shenyang, 10-14.