Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Tomáš Knopp

On the Possibility of ESP Data Use in Natural Language Processing

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Barbora Vidová Hladká, Ph.D.
Studijní program: Informatika
Studijní obor: Obecná informatika

2011

I would like to sincerely thank my advisor Barbora Vidová-Hladká for her immense support and advice.

Contents

Introduction **6**

1 The ESP Game **7**
- 1.1 Why ESP Game? . . . 7
- 1.2 ESP Game Session . . . 7
- 1.3 Agreement on a Label by Two Individual Players . . . 9

2 Coreference Resolution **10**
- 2.1 Introduction and Motivation . . . 10
- 2.2 Coreference . . . 10
- 2.3 Data . . . 11
  - 2.3.1 Analysis of the English Data . . . 11
  - 2.3.2 The Retrieval of Czech Data and Their Analysis . . . 12
- 2.4 Tools . . . 15
  - 2.4.1 Label Viewer Application . . . 15
  - 2.4.2 Algorithm for Finding the Pseudo-Coreference Chains . . . 16
  - 2.4.3 Chains Viewer application . . . 20
  - 2.4.4 Coreferences and Pseudo-coreferences on the Chains Viewer screenshot . . . 21
- 2.5 Evaluation . . . 22
  - 2.5.1 Evaluation Data Description . . . 22
  - 2.5.2 Comparison of Czech versus English data . . . 23
- 2.6 Conclusion . . . 23

3 Lexical database WordNet and the ESP Data **24**
- 3.1 What is WordNet . . . 24
- 3.2 The Possible Benefits of the ESP Data to the WordNet . . . 24
- 3.3 Introduction to (Czech) WordNet . . . 25
  - 3.3.1 Definitions . . . 25
  - 3.3.2 Definition of WordNet . . . 25
  - 3.3.3 The general types of relations between WordNet synsets according to Wikipedia . . . 26

3.4 The Czech WordNet database format 27
3.4.1 Description of the database file 27
3.4.2 An Excerpt from the Database File 28
3.5 WordNet versus ESP data 28
3.5.1 Introduction 28
3.5.2 Computing the Synonymity of a Set of Labels 29
3.5.3 Definition of method 30
3.5.4 Results of the Czech WordNet versus Czech ESP Data . . . 30
3.5.5 Results of the English WordNet versus (English) ESP Data . 34
3.5.6 Conclusion 34

4 Conclusion **38**
4.1 Benefits of the ESP Data to the WordNet Database 38
4.2 ESP Data Use for Automatic Coreference Resolution 38
4.3 Possible Future Improvement and Alternatives 39

References **55**

Název práce: On the Possibility of ESP Data Use in Natural Language Processing
Autor: Tomáš Knopp
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Mgr. Barbora Vidová Hladká, Ph.D.
E-mail vedoucího: hladka@ufal.mff.cuni.cz

Abstrakt: Cílem této bakalářské práce je prozkoumat databázi popisků obrázků ze hry ESP z pohledu zpracování přirozeného jazyka. ESP hra je online hra, ve které lidští hráči konají užitečnou práci - obrázky popisují. Výstupem hry ESP je pak databáze obrázků a jejich popisků. Zajímá nás, zda by data nashromážděná v průběhu získávání popisků obrázků mohla být užitečná pro úkoly zpracování přirozeného jazyka. Konkrétně máme na mysli úkol automatického určování koreferencí, rozšíření lexikální databáze WordNet, zjišťování idiomů a zjišťování slovních spojení. V této bakalářské práci se zaměříme na první dva z nich, tedy na použití databáze pro automatické určování koreferencí a na prozkoumání jejího potenciálního přínosu lexikální databázi WordNet.

Klíčová slova: hra ESP, popisky obrázků, texty vs. obrázky

Title: On the Possibility of ESP Data Use in Natural Language Processing
Author: Tomáš Knopp
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Barbora Vidová Hladká, Ph.D.
Supervisor's e-mail address: hladka@ufal.mff.cuni.cz

Abstract: The aim of this bachelor thesis is to explore this image label database coming from the ESP game from the natural language processing (NLP) point of view. ESP game is an online game, in which human players do useful work - they label images. The output of the ESP game is then a database of images and their labels. What interests us is whether the data collected in the process of labeling images will be of any use in NLP tasks. Specifically, we are interested in the tasks of automatic coreference resolution, extension of the lexical database WordNet, idiom detection, and collocation detection. In this bachelor thesis we deal with the first two of them, which is the task of the automatic coreference resolution and the task of exploring the potential benefits to the lexical database WordNet.

Keywords: ESP game, image labels, texts vs. images

Introduction

The database of the English labels we work with is an output of the extremely popular on-line game called the *ESP game*.[1] Figure 1 and a list of words *london*, *man*, *driver*, *red*, *car*, *people*, *england*, *decker*, *double decker*, *transportation*, *ride*, *double*, *tour*, *road* demonstrate a sample pair of image and its labels that represent the output of selected ESP game sessions.

Figure 1: The ESP image 1

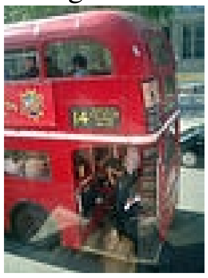
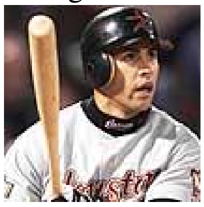


Figure 2 and a list of words *guy*, *hat*, *swing*, *uniform*, *man*, *game*, *sport*, *red*, *hit*, *bat*, *ball*, *houston*, *black*, *face*, *mouth*, *sports*, *white*, *helmet*, *team*, *astros*, *hitter*, *play*, *player* is another example of an image - label pair from the ESP game.

The aim of this bachelor thesis is to explore this image label database from the natural language processing (NLP) point of view. What interests us is whether the data collected in the process of labeling images will be of any use in NLP tasks. Specifically, we are interested in the tasks of coreference resolution, WordNet extension, idiom detection, and collocation detection. In this bachelor thesis we deal with the possible use of the image label database for coreference resolution and WordNet extension.

Figure 2: The ESP image 2



[1] `http://gwap.com`

Chapter 1

The ESP Game

1.1 Why ESP Game?

The ESP game is basically an online game created for generating and harvesting valuable descriptions of general images. Getting good description, i.e. good labels is a hard task for computers. On the other hand, if a human sees some picture, s/he immediately knows a few expressions, that describe it well. Simply put, the game uses human players' intelectual capacity for labeling images. The usual problem is that one usually has to pay people for performing such a laborous work. However, thanks to the ESP game people do the work voluntarily and for free, because they enjoy the game.

If we realize that the game actually performs a computation because through the game the players assign labels to images, then we can see the individual players as processors and the specifically designed game as an algorithm. The design of the game guarantees that players "compute" well. The ESP game is thus an example of a human algorithm game, which is a term coined by Luis von Ahn in (von Ahn, 2005).

1.2 ESP Game Session

A game session is played by two randomly chosen players. There is a time limit for the session. During the session both the players are presented with the same series of pictures. Both players always see the same image and when seeing a particular picture, their task is to try enter such string, that they think their partner is entering. While playing none of the partners is able to see their partner's guesses. The partners do not know each other and they are unable to communicate. So the task is they should try thinking as their partner would have and the ESP stands for "extrasensory perception" here.

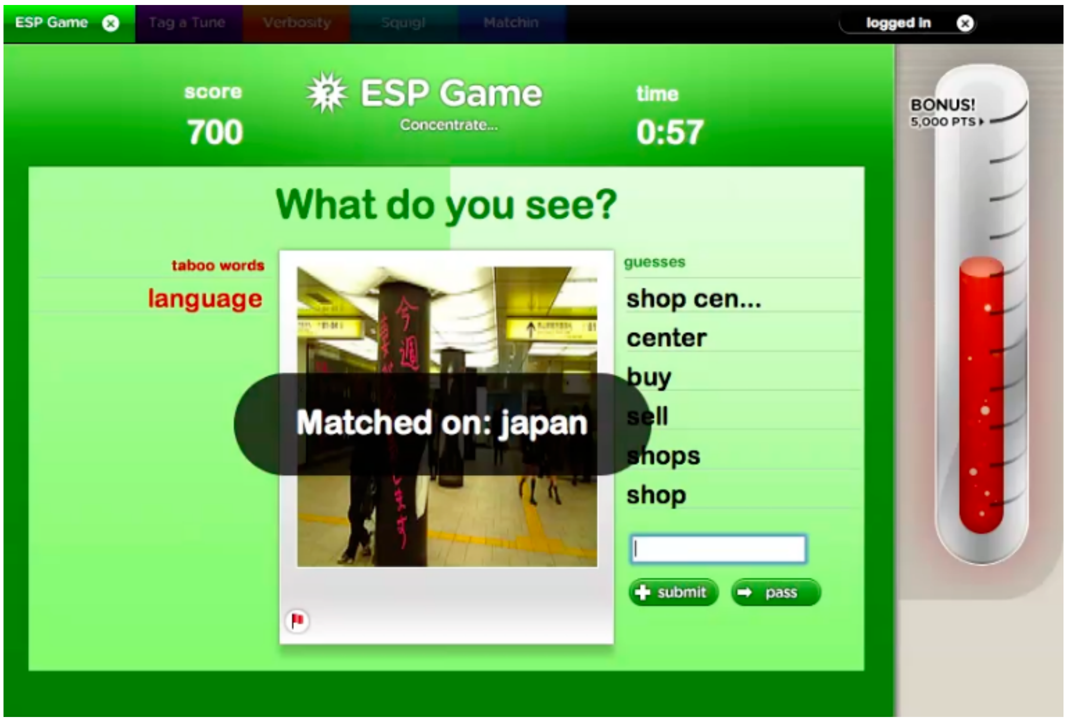


Figure 1.1: The player's online interface. Players have just agreed on a string "japan".

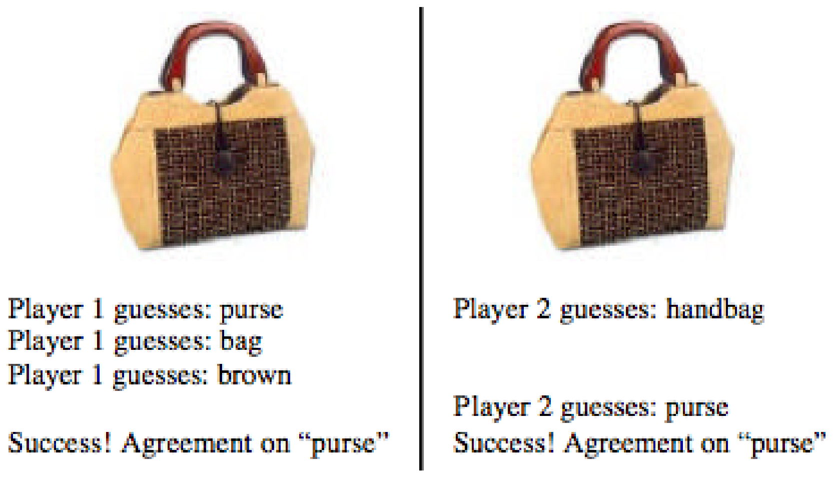


Figure 1.2: ESP Game session: players agreeing on a string

1.3 Agreement on a Label by Two Individual Players

If the two players enter the same strings when presented with the picture at some time during which the picture is shown, they are given points for this agreement. Both the players have to agree on a particular string so that the string becomes a label for the image since agreement by a pair of independent players implies that the string (consequently label) is probably meaningful. It turns out that the task to write the same string as the partner guarantees (under the game's conditions) that the strings upon which there were agreements are meaningful description to their images. It means that such strings are true labels of their images. The Figure 1.2 is ilustrating an agreement on a string. It comes from (von Ahn, Dabish, 2004).

Each image is associated with a list of taboo words that are not allowed to be entered by the players: the string becomes a label when two players agree on it and consequently the label becomes a taboo word associated with the image, which will be used when the system reuploads the image into a new game session. We are not familiar with any other restriction put on the strings which players write during the game but their lengths - up to 13-character-long strings are allowed. A *label* can be a single-word label (*london*) or a multi-word label (*double decker*).

Chapter 2

Coreference Resolution

2.1 Introduction and Motivation

In this chapter, we are focusing on the task of automatic coreference resolution; the issue we tackle is whether the image labels can be a help for this task. We speculate that there can be labels among the image labels that co-refer, i.e. they refer to the same entity. We have not found any paper that addresses the same issue so far. So not only given that, it is almost impossible to predict the results before performing experiments.

This chapter is organized as follows: we remind the notion of coreference in Section 2.2. In Subsection 2.3.1 we present statistics on the ESP game image labels. The application designed for user-friendly viewing the labels and their relations is introduced in Subsection 2.4.1. The algorithm to construct pseudo-coreference chains in the text is provided in great details in Subsection 2.4.2. The application for viewing and comparing the coreference versus pseudo-coreference chains is described in Subsection 2.4.3. Evaluation of pseudo-coreference chains against manual annotation is discussed in Section 2.5. We conclude with Section 2.6.

2.2 Coreference

Coreference occurs when several referring expressions in a text refer to the same entity (e.g. person, thing, fact). A *coreferential pair* is a pair of the referring expressions. A sequence of coreferential pairs referring to the same entity in a text forms a *coreference chain*. In the passage from (Doyle, 1887), one can read the following coreference chain *I, I, me, I, me man*; another coreference chain is *someone, Stamford, who, dresser* can be seen there: *On the very day that **I** had come to this conclusion **I** was standing at the Criterion Bar, when someone tapped **me** on the shoulder, and turning round. **I** recognized young Stamford who had been a dresser* *under* ***me*** *at Barts. The sight of a friendly face in the great wilderness of London is a pleasant thing indeed to a lonely* ***man****.*

2.3 Data

2.3.1 Analysis of the English Data

The *ESPgame100k* data (”data”)[1] we inspect throughout our project come from the ESP Game (von Ahn, Dabish, 2004)[2] as described in Chapter 1

As a result of the ESP game sessions, there is a label-set for each of the images that have already been assigned some labels in the game and these label-sets are stored in the game's database together with the corresponding image. The data, which we use as the input for our project, is a sample part of the whole database that consists of 100,000 image-labels pairs.[3]

Thus for each image in the data there is a non-empty label-set. A *label-set* is a set of labels related to the same image, like *car, decker, double, double decker, driver, england, london, man, people, red, ride, road, tour, transportation.*

Neighbors of a label lab is a set of labels which includes all labels from all label-sets in which the label lab occurred. So the neighbors of *double decker* include all labels from the label-set from Figure 1 including lab itself, i.e. *car, decker, double, double decker, driver, england, london, man, people, red, ride, road, tour, transportation*, but the neigbors of lab also include labels from other label-sets, for instance *cloud, tire, wheels* because each one of them co-occur together with lab in at least one label-set. So we can say that for instance *wheels* is a neigbor of lab.

$freq(w_1, w_2)$ stands for the number of label-sets where the labels w_1, w_2 co-occur together. We call it `neighborhood frequency`. Table 1 provides statistics acquired from the data, i.e. from the ESPgame100k sample.

According to the game designers, the game server is equipped with an English dictionary to alert players when they have misspelled a word. However, we discover that not always is the label a good description of the particular image. It can be misspelled, which is frequent, or it even does not have to be English. Some examples of the various spelling errors are *havent* or *family fued*. Examples of non-English labels are *zeitungen*, *nuestra* or *zukunft*. We use the open source spell checker `Aspell`[4] to discover such errors. However, we do not have to handle them in a special manner since our procedure to construct pseudo-coreference chains in texts does not take them into account anyway.

[1] `http://www.cs.cmu.edu/~biglou/resources/`

[2] `http://gwap.com`

[3] We do not know exactly what good label threshold X was used for the images presented in the data. According to literature (for ex. (von Ahn, 2005)), we suppose $X = 1$.

[4] `http://aspell.net/`

# of unique label-sets	100,000
average # of labels in label-sets	14
the biggest label-set size	42
the smallest label-set size	8
# of unique labels	29,845
# of unique single word labels	27,602
# of unique multi word labels	2,243
# of unique neighbors	1,676,856

Table 2.1: The ESPgame100k data: statistics

We also tagged the labels with `Stanford Log-linear Part-Of-Speech Tagger` (Toutanova, Klein, Manning, Singer, 2003) to discover foreign words.

2.3.2 The Retrieval of Czech Data and Their Analysis

We translate the original data from English to Czech using `The Czeng Probability Dictionary`[5] ("dictionary"). For a given English word and its part of speech tag (POS tag) the dictionary contains different Czech words with POS tags and with probabilities P that the Czech word and its POS tag is a good translation for the English word and its POS tag. We call these probabilities translation probabilities. As you can see in Table 2.2, the Czech translations are sorted by the translation probabilities P.

The Algorithm for Translating Label-sets into Czech

English label-sets are translated into Czech label by label.

One English label can bear more morphological meanings mainly because of the common conversion in English ("run" as a verb or noun). Because the labels are not part of a broader context like sentence, we cannot say for sure which POS the original label is. Since we want to preserve as many possible meanings of the original label as possible in the translated label-set, but still we do not want the translated label-sets to be too large either, we chose to translate the original label together with the POS tag.

In the dictionary, we look up the English label lab that we want to translate into Czech. The label might appear in the dictionary with more POS tags as a different part of speech. In Table 2.3 we can see an example of looking up the label *twelve*.

[5]It is a dictionary that was extracted from the parallel corpus CzEng by Zdeněk Žabokrtský at UFAL, Charles University in Prague in 2008.

English	Czech	P
Aggie#N	hnojárna#N	1.000000
Agnes#N	Agnes#N	0.396889
Agnes#N	Anežka#N	0.283199
Agnes#N	škola#N	0.240103
Agnes#N	fond#N	0.079809
Agreements#N	dohoda#N	0.819628
Agreements#N	přidružení#N	0.094835
Agreements#N	praktika#N	0.025373
Agreements#N	uzavřený#A	0.021605
Agreements#N	restriktivní#A	0.017228
Agreements#N	odvětví#N	0.012684
Agreements#N	evropský#A	0.008648
AgroSciences#N	společnost#N	0.502078
AgroSciences#N	AgroSciences#N	0.497922

Table 2.2: Example of the dictionary format

When translating *twelve* in this example we get one translation for *twelve* as a cardinal number and another translation for *twelve* as a noun.[6]

To translate the combination of label and a POS tag we always pick the most probable Czech translation without further investigations. In our case the translations are *dvanáct* and *dvanáctka* and they are highlighted in Table 2.3.

When translating the label-set into Czech, we simply substitute the English label with a list of Czech translations as described in the previous step. The English multi-word labels are currently not translated, unless they appear in the `Czeng dictionary` as a whole expression.[7]

Example: Translating Label-sets

In this example we illustrate the translation in which the English label-set *{asian, background, bag, blue, desk, desktop, face, girl, keyboard, kid, moni-*

[6]It would also be possible to translate only the one combination of label and its POS tag which has the highest translation probability. This would obviously ensure that we always get only one translation for a label. In the example it would be *dvanáct*, because $P(\textit{dvanáct}) > P(\textit{dvanáctka})$. However, it would also mean that we would not take the other part of speech types into account at all. As we want to keep as many morphological meanings as possible, we better decided to have an English label translated by more Czech labels, than to lose meanings when choosing only the most probable part of speech.

[7]Another method would be to tear apart the English multi-word labels, translate them word by word and then put them together. But this has no sense considering that the algorithms we test on the ESP data work in fact only with single-word labels.

English	Czech	P
twelve#C	**dvanáct**#C	0.845118
twelve#C	jeden#C	0.032349
twelve#C	dvanáctiměsíční#A	0.024747
twelve#C	půlnoc#N	0.022146
twelve#C	půl#N	0.016610
twelve#C	rok#N	0.013889
twelve#C	dvanáctý#C	0.012131
twelve#C	dvanáctiletý#A	0.008286
twelve#C	dvanáctery#C	0.008087
twelve#C	ještě#D	0.005557
twelve#C	pryč#D	0.005543
twelve#C	poledne#N	0.005537
twelve#N	**dvanáctka**#N	0.316118
twelve#N	dvanáct#N	0.161365

Table 2.3: Looking up the label *twelve* in the dictionary

tor, mouse, pc, picture, red, screen, smile, wallpaper, white, windows, woman} is translated into the Czech label-set *{asijský, Asie, pozadí, společenský, pytel, sebrat, modrý, stůl, plocha, čelit, tvář, dívka, holkařit, klávesnice, legrace, zhýčkaný, dítě, sledovat, monitor, pohyb, myš, PC, obrázek, představovat, věrná, červený, methylčerveně, prověrka, obrazovka, usmát se, úsměv, tapetovat, tapeta, bílý, bezvousý, okna, žena}*.

Table 2.4 shows parts from the dictionary used for translating the individual English labels from the label-set into Czech. This excerpt is shortened. For a combination of a label and its POS tag only the most probable Czech translation (the one actually used for the translation of a label and its POS tag) is shown here and the other translations were ommited from this excerpt of the dictionary.

The first label in the original label-set is *asian*. So the algorithm looks it up in the dictionary and finds there are multiple translations for *asian* as an adjective and multiple translations for *asian* as a noun (only the most probable of them are displayed in Table 2.4). The algorithm takes the most probable translation for *asian* as a noun and as an adjective, i.e. the two words *asijský* and *Asie*. They both become the translation for the original label *asian*.

In the same fashion, *background* as an adjective is translated into *společenský* and *background* as a noun is translated into *pozadí*; *white* as an adjective and as a noun are translated into *bílý* and *white* as a verb is translated into *bezvousý*.[8]

[8]Of course, this is a nonsense. It only prooves that the dictionary is not flawless, because it had been extracted by a program from the CzEng parallel corpus.

English label	Czech label	P
asian#A	asijský#A	0.750789
asian#N	Asie#N	0.389752
background#A	společenský#A	1.000000
background#N	pozadí#N	0.803533
bag#N	pytel#N	0.219575
bag#V	sebrat#V	0.275134
...	...	...
white#A	bílý#A	0.877069
white#N	bílý#A	0.614535
white#V	bezvousý#A	0.540322
windows#N	okna#N	1.000000
woman#N	žena#N	0.906362

Table 2.4: The excerpt from the dictionary used for translating the label-set. Shortened.

2.4 Tools

2.4.1 Label Viewer Application

The `Label Viewer` is an application designed for user-friendly viewing the English or Czech labels and their relations. It has three main parts as depicted in Figure 2.1:

- **part 1 - list of labels**. Labels from the ESP data are displayed here. Each label has its `id` and `frequency`. Frequency means the number of label-sets containing the label. The label *cathedral* is selected in part 1 of the example.

- **part 2 - list of label-sets**. Selecting a label from part 1 reveals the list of id's of label-sets in which the label (i.e. *cathedral*) occurred (part 2 - left column). One can click on a label-set id to inspect the label-set's labels (part 2 - right column) and the original picture which belongs to the viewed label-set. In the Figure 2.1 the label-set with $id = 493$ is shown.

- **part 3 - list of neighbors**. In part 3, the neighbors for a selected label (i.e. *cathedral*) are shown in the `neighbors` column. The `frequency` here is the number of label-sets in which the selected label (*cathedral*) and the selected neighbor (*spire* in the right column in part 3) co-occurred together, i.e. $freq(w_1, w_2)$ described in 2.3.1. In our example 56 means that labels *cathedral* and *spire* co-occured together in 56 different label-sets.

Both lists of labels and neighbors can be resorted alphabetically or by frequency. By default, labels are sorted alphabetically and neighbors by frequency.

2.4.2 Algorithm for Finding the Pseudo-Coreference Chains

We take an input text and we find the pairs of 'semantically related' ESP labels in this text. We call the pairs of ESP labels in the texts *pseudo-coreference pairs*. Then we construct *pseudo-coreference chains* from them.

We know how many times two labels co-occur together in the data, i.e. how many times they label the same images. We measure the degree of being semantically related labels by setting a `neighborhood threshold` Z. Both the pseudo-coreference pairs and chains are depending on the variable `neighborhood threshold` Z.

For a given Z, we search for the pseudo-coreference chains in these five steps. The algorithm is ilustrated with on an example.

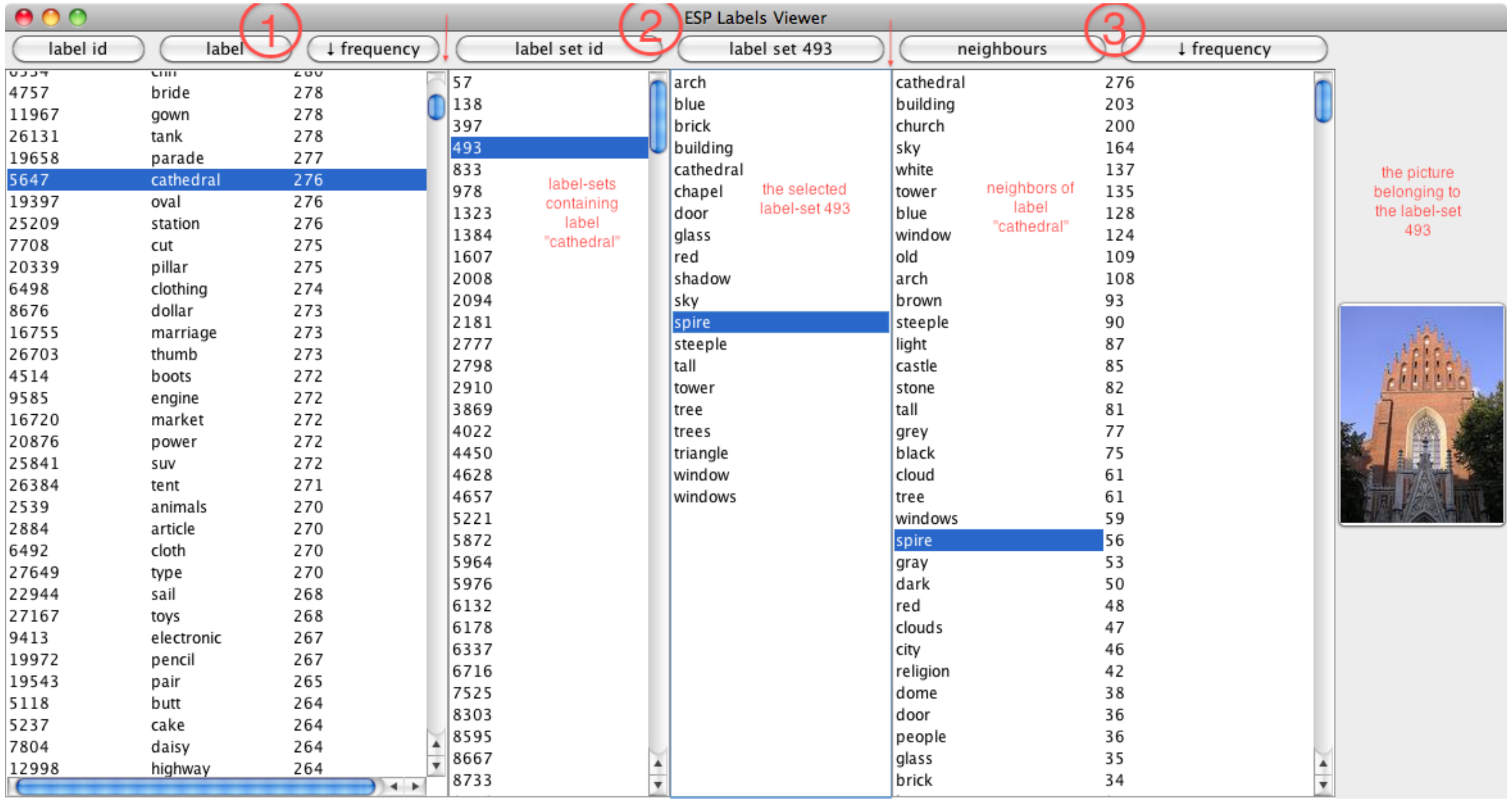
1. We have `Penn Treebank POS tags`[9] for each word in the text. Since we suppose that the candidates for coreference are *Wh-determiners*, *Wh-pronouns*, *nouns*, *personal pronouns* and *cardinal numbers*, the words with the `Penn Treebank POS tags WDT, WP, N, PRP, CD`, respectively, become the candidates to be members of coreferential chains; thus we unlock them for the next steps while we lock the words with other POS tags.

   In this example of input text, the unlocked words are highlighted. ...″There was a modern brown **building** with an old antique **arch** above the **door.** The **arch** didn′t fit to the **design** of the **building** so much that **I** thought the **architect who** projected it must have been mad to use **it** here.″...

2. On the ESP data layer, we iterate through label-sets and take each label-set's labels as nodes V of a graph $G = (V, E)$ in which edges E are defined so: $e = (v_1, v_2) \in E$ if $freq(v_1, v_2) \geq Z$. We call this graph `neighborhood graph`.

   In Table 2.5 there are label-set's neighborhood frequencies, from which we construct the `neighborhood graph` as described above (see the resulting neighborhood graph in Table 2.6 and in Figure 2.2). Because neighborhood of labels is a symetrical relation, the neighborhood graph is undirected.

[9]http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html


ESP Labels Viewer
1
2
3
label-sets containing label "cathedral"
the selected label-set 493
neighbors of label "cathedral"
the picture belonging to the label-set 493

label id	label	↓ frequency
4757	bride	278
11967	gown	278
26131	tank	278
19658	parade	277
5647	cathedral	276
19397	oval	276
25209	station	276
7708	cut	275
20339	pillar	275
6498	clothing	274
8676	dollar	273
16755	marriage	273
26703	thumb	273
4514	boots	272
9585	engine	272
16720	market	272
20876	power	272
25841	suv	272
26384	tent	271
2539	animals	270
2884	article	270
6492	cloth	270
27649	type	270
22944	sail	268
27167	toys	268
9413	electronic	267
19972	pencil	267
19543	pair	265
5118	butt	264
5237	cake	264
7804	daisy	264
12998	highway	264

label set id
57
138
397
493
833
978
1323
1384
1607
2008
2094
2181
2777
2798
2910
3869
4022
4450
4628
4657
5221
5872
5964
5976
6132
6178
6337
6716
7525
8303
8595
8667
8733

label set 493
arch
blue
brick
building
cathedral
chapel
door
glass
red
shadow
sky
spire
steeple
tall
tower
tree
trees
triangle
window
windows

neighbours	↓ frequency
cathedral	276
building	203
church	200
sky	164
white	137
tower	135
blue	128
window	124
old	109
arch	108
brown	93
steeple	90
light	87
castle	85
stone	82
tall	81
grey	77
black	75
cloud	61
tree	61
windows	59
spire	56
gray	53
dark	50
red	48
clouds	47
city	46
religion	42
dome	38
door	36
people	36
glass	35
brick	34

Figure 2.1: The screenshot of the label viewer program

$Z = 500$	arch	brown	building	cathedral	church	door	old	wood
arch	718	213	418	108	235	232	156	66
brown		11081	1148	93	293	528	1056	1482
building			6915	203	753	771	836	364
cathedral				276	200	36	109	14
church					1187	172	297	69
door						2248	317	309
old							5866	318
wood								3056

Table 2.5: Neighbor frequencies in a label-set.

$Z = 500$	arch	brown	building	cathedral	church	door	old	wood
arch	1	0	0	0	0	0	0	0
brown		1	1	0	0	1	1	1
building			1	0	1	1	1	0
cathedral				0	0	0	0	0
church					1	0	0	0
door						1	0	0
old							1	0
wood								1

Table 2.6: Neighborhood graph of a label-set given by matrix.

3. For each neighborhood graph we start the algorithm for finding connected components (Hopcroft, Tarjan, 1973). This algorithm computes the neighborhood graph's components: $G_1 = (V_1, E_1), ...G_n = (V_n, E_n)$. These components are the costituting units of pseudo-coreference chains.

   The graph in Figure 2.2 has three components which are defined by these sets of vertices $V_1 = \{arch\}$, $V_2 = \{brown, building, church, door, old, wood\}$ and $V_3 = \{cathedral\}$.

4. We now have to go down from the layer of labels to the layer of the input text. A label can appear in the text $0, ..., n_{label}$ times. So for a neighborhood graph we take each of its components $G_i = (V_i, E_i)$ from the previous step and propagate the labels from that component G_i to the textual layer. For each vertex v (i.e. label) we save the relevant occurences of the label v as a word in the input text. If label v is not present in the text, we save nothing, if it is present $1, ..., n_v$ times, we save only the relevant occurences (i.e. those that have been unlocked in the first step of the algorithm). So as a result for each

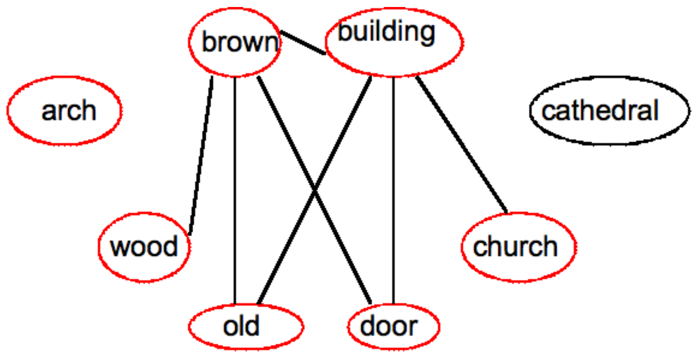


Figure 2.2: Neighborhood graph of a label-set

of the component $G_i = (V_i, E_i)$ of the neighborhood graph we get a list of occurences (unlocked words) of the vertexes V_i (i.e. labels) in the input text.

The resulting lists from the example are $List_1 = \{arch, arch\}$ and $List_2 = \{building, building, door\}$.

5. We filter out lists of occurences with size ≤ 1. The last step is to resort the lists of occurences, so that the pseudo-coreference chains are well ordered. A list of words is well ordered, if for the list $l = w_1, ..., w_n$ it is true that $\forall i = 1, ..., i : pos(w_i) < pos(w_{i+1})$, where $pos(w_i)$ is the function that returns the word's position within the input text. The sorted lists with size ≥ 2 are the `pseudo-coreference chains`.

   Both the lists from the previous step contain at least two words, and after ordering we get these pseudo-coreference chains: $Chain_1 = \{arch, arch\}$ and $Chain_2 = \{building, door, building\}$.

Summing up the Algorithm

1. Choose a threshold Z and iterate through label-sets.
2. Take the labels from the current label-set ($lset$) as nodes of a graph, where two labels (i.e. vertex) $lab1, lab2$ are adjacent if $freq(lab1, lab2) \geq Z$ (so called `neighborhood graph`).
3. Find the connected components of the neighborhood graph. Iterate through those components.
4. For labels of a component c find unlocked occurences of those labels in the input text. These well ordered occurences (if there are at least 2 of them) form a pseudo-coreference chain.

Computing the pseudo-coreference chains - time complexity

There are $n = 100,000$ label-sets in the data, let the biggest label-set's size be max. $freq(lab1, lab2)$ can be determined in $O(1)$ (the whole neighborhood relation is initialized only once after the program starts). For each label-set we construct the adjacency matrix in $max^2 * O(1) = O(1)$. The algorithm for finding connected components is linear in the number of vertex, which has upper bound max. So for each label-set, we construct the matrix in $O(1)$ and then in $O(max) = O(1)$ we find the connected components. So finally we get linear time $O(n)$ in the #label-sets in the data to count all the connected components.

Now getting to the textual layer, suppose that there are $m \leq max$ different labels in a particular label-set that have to be expanded into pseudo-coreference chains on the textual layer.

Let the text length be k. Then each label can theoretically appear on k positions. So while propagating to the textual layer, we get to much worse complexity class. $k * k ... * k = k^{max}$, which yields exponential time $O(k^{max})$ for propagating a label, resp. a connected component from a label-set (measured by the length of the input text). The whole time complexity of counting the pseudo-coference chains for data with n label-sets and maximum size of the label-set max is thus $O(n * k^{max})$.

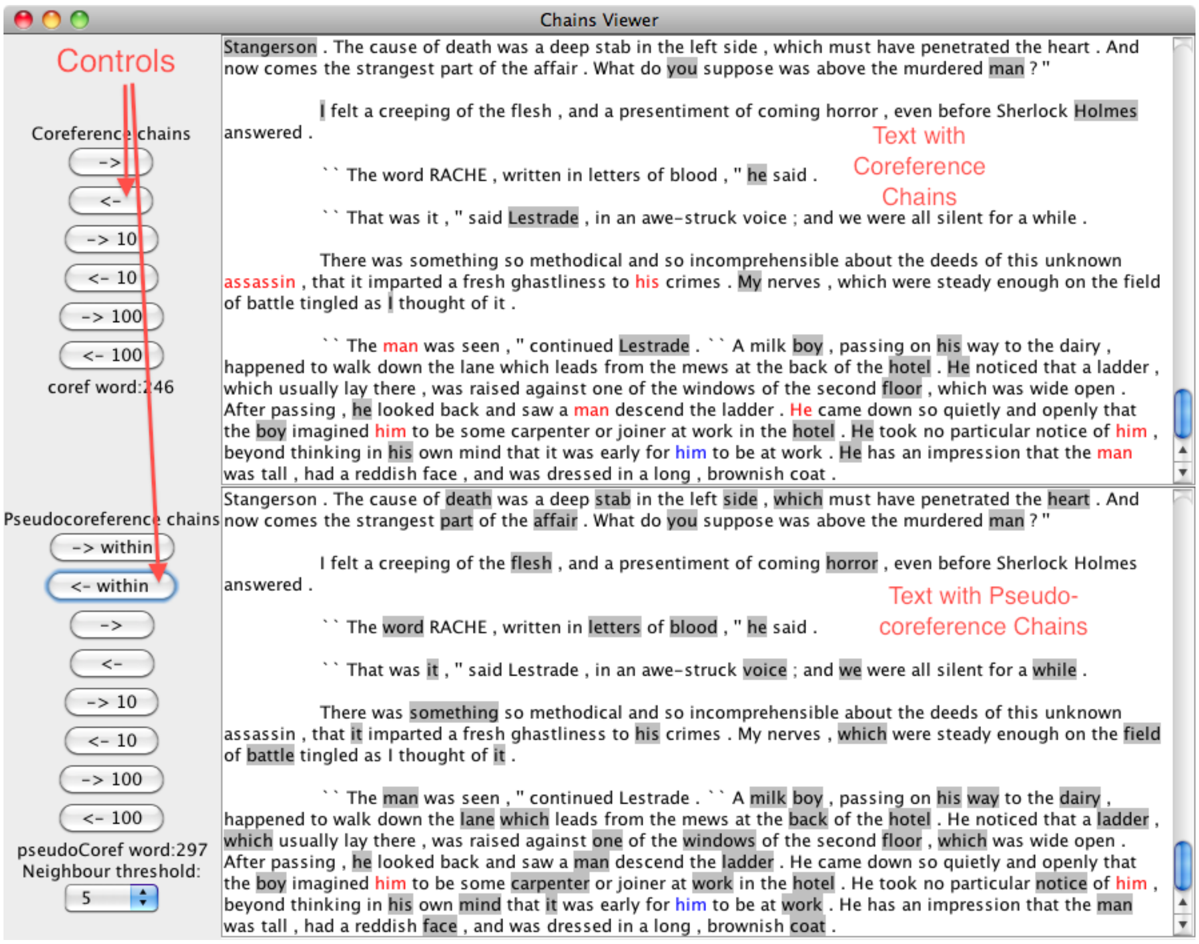
This does not matter in this project, because here we were testing the algorithm on very small input texts where only excerpts of a length of 100 sentences were used. The exponential complexity here is due to not having a limit or restriction on how far from each other two pseudo-coreference words can be. Here the domaine used is the whole text. If our method turned out to be of any use, it would be simple to introduce such rule and limit the domaine to e.g. a paragraph, which size is presumably not dependent on the length of the input text and can be taken as a constant. If we limit the size of paragraph in the input text to $para$, then the resulting time complexity is linear $O(n * para^{max}) = O(n)$.

2.4.3 Chains Viewer application

The *Chains Viewer* program is an application designed for computing the pseudo-coreference chains over a sample Czech or English text in the csts[10] format based on the ESP data. It compares the pseudo-coreference chains against the manually annotated coreference chains and lets user browse through the pseudo-coreference and coreferential chains. The screenshot of the program is in Figure 2.3.

The program consists of two parts. The controls, which allow the user to browse through the coreferences and pseudo-coreferences and the display, which shows the coreferences and pseudo-coreferences within the text. The controls are located on

[10] Czech Sentence Tree Structure is a format developed at ÚFAL, Charles University in Prague for Czech National Corpus and Prague Dependancy Treebank. `https://wiki.ufal.ms.mff.cuni.cz/format-csts`



coref word:246

pseudoCoref word:297
Neighbour threshold: 5

Stangerson . The cause of death was a deep stab in the left side , which must have penetrated the heart . And now comes the strangest part of the affair . What do you suppose was above the murdered man ? "

I felt a creeping of the flesh , and a presentiment of coming horror , even before Sherlock Holmes answered .

`` The word RACHE , written in letters of blood , " he said .

`` That was it , " said Lestrade , in an awe-struck voice ; and we were all silent for a while .

There was something so methodical and so incomprehensible about the deeds of this unknown assassin , that it imparted a fresh ghastliness to his crimes . My nerves , which were steady enough on the field of battle tingled as I thought of it .

`` The man was seen , " continued Lestrade . `` A milk boy , passing on his way to the dairy , happened to walk down the lane which leads from the mews at the back of the hotel . He noticed that a ladder , which usually lay there , was raised against one of the windows of the second floor , which was wide open . After passing , he looked back and saw a man descend the ladder . He came down so quietly and openly that the boy imagined him to be some carpenter or joiner at work in the hotel . He took no particular notice of him , beyond thinking in his own mind that it was early for him to be at work . He has an impression that the man was tall , had a reddish face , and was dressed in a long , brownish coat .

Stangerson . The cause of death was a deep stab in the left side , which must have penetrated the heart . And now comes the strangest part of the affair . What do you suppose was above the murdered man ? "

I felt a creeping of the flesh , and a presentiment of coming horror , even before Sherlock Holmes answered .

`` The word RACHE , written in letters of blood , " he said .

`` That was it , " said Lestrade , in an awe-struck voice ; and we were all silent for a while .

There was something so methodical and so incomprehensible about the deeds of this unknown assassin , that it imparted a fresh ghastliness to his crimes . My nerves , which were steady enough on the field of battle tingled as I thought of it .

`` The man was seen , " continued Lestrade . `` A milk boy , passing on his way to the dairy , happened to walk down the lane which leads from the mews at the back of the hotel . He noticed that a ladder , which usually lay there , was raised against one of the windows of the second floor , which was wide open . After passing , he looked back and saw a man descend the ladder . He came down so quietly and openly that the boy imagined him to be some carpenter or joiner at work in the hotel . He took no particular notice of him , beyond thinking in his own mind that it was early for him to be at work . He has an impression that the man was tall , had a reddish face , and was dressed in a long , brownish coat .

Figure 2.3: The screenshot of the Chains Viewer program

the left side of the program and the display is located in the central part of the program.

The controls and display are further divided into two parts. The upper part deals with coreference chains and the lower part with pseudo-coreference chains.

2.4.4 Coreferences and Pseudo-coreferences on the Chains Viewer screenshot

In the upper display of the Chains Viewer program's screenshot there is an example of a coreference. The coreference shown is `{assassin, his, man, man, He, him, him, him, man}`.

In the lower display where pseudo-coreferences are shown, one can read this pseudo-coreference: `{him,him,him}`.

2.5 Evaluation

We have selected *A Study in Scarlet* by Sir Arthur Conan Doyle (Doyle, 1887) with manual annotation of coreference. For the purpose of our evaluation, we work with a 100-sentence part of it. For different Zs, we calculate how much pseudo-coreference chains overlap with coreference chains. As quantitative measures, we use standard precision (P) and recall (R). The results for three chosen Zs are listed in Table 2.7 and Table 2.8.

Z	# of p.-c. chains	R (%)	P (%)
5	7,129	10	0.06
100	5,146	10	0.08
500	2,347	10	0.17

Table 2.7: Pseudo-coreference chains in *A Study in Scarlet* - no stems used

Z	# of p.-c. chains	R (%)	P (%)
5	1,073	2.04	0.09
100	582	2.04	0.17
500	176	2.04	0.57

Table 2.8: Pseudo-coreference chains in *Studie v Šarlatové* - no stems used

2.5.1 Evaluation Data Description

A Study in Scarlet / *Studie v Šarlatové* are texts which were manually annotated. The inner format of the texts which we work with is *csts*[11]. The Czech text is the translation of the English text. Both the texts contain among others coreferences in the text and the stem forms and POS tags for the individual words in the text.

We started to evaluate the pseudo-coreference chains algorithm on the English text. We chose an excerpt from the text, which includes 100 sentences from the text. First we started with the very first 100 sentences from *A Study in Scarlet* but it turned out that in this excerpt, the proportion of the pronouns in the coreferences was too high. So we divided the text into parts formed by 100 sentences each and then counted the percentage of pronouns in the coreferences for each part. The

[11]`http://tinyurl.com/6h9mv5w`

lowest percentage had the part formed by the part starting with 1200th sentence and ending with 1299th sentence. So this is the final excerpt we use in this project for the evaluation of the pseudo-coreference chains algorithm.

When shifting to the evaluation of the algorithm over the Czech text, we had to find the matching counterpart of the chosen English excerpt. This was done manually, because due to the translation process the sentence id's are not matching between the Czech and English text.

2.5.2 Comparison of Czech versus English data

We can see, that running the algorithm over the Czech text yields approximately 10 times less pseudo-coreference chains than over the English text. This is most probably due to the inflective character of the Czech language and the synthetic character of the English language and the fact, that we did not take into account the word stem forms.

2.6 Conclusion

We can see that by far the results are not statistically significant. The proportion of pronouns in the coreference chains is very high with respect to the fact that the pronouns appear in the image-label database rarely. It motivates us to select for future experiments a text of a different genre. Although the coreference and pseudo-coreference chains often overlap, seldom does a pseudo-coreference chain fully correspond to a coreference chain. So the ESP data is not suitable for finding the coreference chains.

Chapter 3

Lexical database WordNet and the ESP Data

3.1 What is WordNet

”WordNet® is a large lexical database of English, developed under the direction of George A. Miller (Emeritus). Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download. WordNet’s structure makes it a useful tool for computational linguistics and natural language processing.” [1]

3.2 The Possible Benefits of the ESP Data to the WordNet

The question posed in this bachelor thesis is: how can be the output of the ESP game used to become a benefit to the WordNet? The WordNet database has been developed for a long time and we do not suppose we could add words or expressions to it based only on simple facts like the neighborhood counts within a given sample of the ESP database. So the only thing we can imagine, if it would be of any use, is to simply add the neighborhood count to the entries within the WordNet database. Because WordNet does not work with individual words, but with synsets, it would be a little harder, but still possible. WordNet could use this extra information as a kind of help when determining which synsets are more closely related than others.

[1] `http://wordnet.princeton.edu/`

For this method to work, it would probably require much bigger portion of the ESP data, then the sample data we have worked with.

3.3 Introduction to (Czech) WordNet

3.3.1 Definitions

Synset - a set of synonyms, e.g. {lak,emailový lak, email}. It bears a concept.
Literal - a particular expression from a synset, e.g. lak.

3.3.2 Definition of WordNet

The Wordnet consists of synsets which are connected by different relations. Because a synset represents a concept, the relations are relations between concepts.

Types of relations in the Czech WordNet

The relations used in the Czech WordNet are *hypernym* (B is a hypernym of A if every A is a (kind of) Y (canine is a hypernym of dog, because every dog is a member of the larger category of canines)), *holo-part*(citronová kůra is a holo part of citron), *near antonym* and *similar to*.

Types of relation in the English WordNet

"Is a" hierarchy

The model used here is known as the "IS A" model. Dog is a canine, citronová kůra is a part of citron, good is a near antonym of evil, etc. As we can see, the synsets are always connected together through the "is a" relation.

WordNet as a graph

The (Czech) WordNet database[2] is in fact a huge (oriented) graph. In this graph the individual vertices are formed by synsets and they are connected by edges of different types which bear the relation between synsets.

SnappyWords.com tool for displaying WordNet online

SnappyWords.com is an internet website which queries the English WordNet database and shows the result in a user-friendly graphical design. An example can be seen in Figure 3.3.2.

2

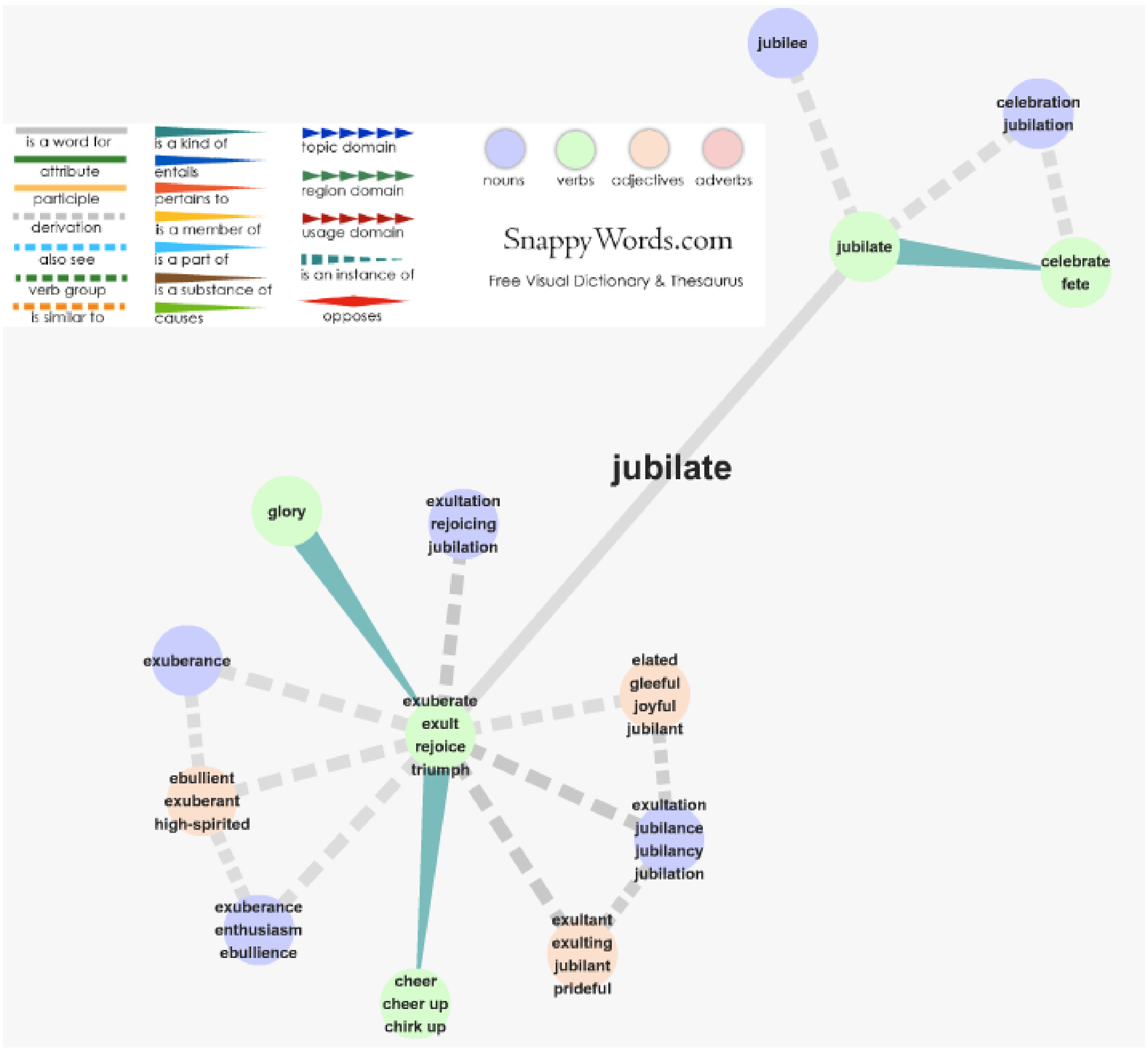


Figure 3.1: Example from SnappyWords.com: WordNet as a graph.

3.3.3 The general types of relations between WordNet synsets according to Wikipedia

[3]

Nouns

hypernyms Y is a hypernym of X if every X is a (kind of) Y (canine is a hypernym of dog, because every dog is a member of the larger category of canines)

hyponyms Y is a hyponym of X if every Y is a (kind of) X (dog is a hyponym of

[3]http://en.wikipedia.org/wiki/WordNet

canine) coordinate terms: Y is a coordinate term of X if X and Y share a hypernym (wolf is a coordinate term of dog, and dog is a coordinate term of wolf)

holonym Y is a holonym of X if X is a part of Y (building is a holonym of window)

meronym Y is a meronym of X if Y is a part of X (window is a meronym of building)

Verbs

hypernym the verb Y is a hypernym of the verb X if the activity X is a (kind of) Y (to perceive is an hypernym of to listen)

troponym the verb Y is a troponym of the verb X if the activity Y is doing X in some manner (to lisp is a troponym of to talk)

entailment the verb Y is entailed by X if by doing X you must be doing Y (to sleep is entailed by to snore)

coordinate terms those verbs sharing a common hypernym (to lisp and to yell)

Adjectives

related nouns
similar to
participle of verb

Adverbs

root adjectives

3.4 The Czech WordNet database format

3.4.1 Description of the database file

The actual database is saved in an xml file. Each line of the file describes a synset. Each line ends with a newline. Description of the synset includes a unique id, part of speech, literals of the synset, the polysemy count (i.e. in how many synsets the literal lies = #senses), the synset's neighbor (if any), the type of the relation to the neighboring synset, and the domain of the concept. Facultatively it contains the definition of the concept, its usage, a note about the synset, time stamp of the person who added the item to the databse, and valency frames of the literals.

3.4.2 An Excerpt from the Database File

```
   <SYNSET> <ID> ENG20-01481708-v </ID> <POS> v
</POS> <SYNONYM> <LITERAL> navíjet <SENSE> 1 </SENSE>
</LITERAL> <LITERAL> natáčet <SENSE> 1 </SENSE>
</LITERAL> </SYNONYM> <ILR> ENG20-01480045-v <TYPE>
hypernym </TYPE> </ILR> <ILR> ENG20-01481921-v
<TYPE> near_antonym </TYPE> </ILR> <STAMP> Karel
2003/12/29 </STAMP> <VALENCY> <FRAME> navíjet, natáčet
kdo1*AG(person:1)=co4*OBJ(line:18) na co4*OBJ(winder:3)
%navíjet drát na cívku </FRAME> </VALENCY> <DOMAIN>
factotum </DOMAIN> </SYNSET>
   <SYNSET> <ID> ENG20-01472715-v </ID> <POS>
v </POS> <SYNONYM> <LITERAL> odpálit <SENSE> 1
</SENSE> </LITERAL> </SYNONYM> <ILR> ENG20-01472314-v
<TYPE> hypernym </TYPE> </ILR> <STAMP> Karel
2003/12/29 </STAMP> <VALENCY> <FRAME> {odpálit}
kdo1*AG(person:1|institution:1)=co4*OBJ(device:1) %NASA
odpálila raketu na Měsíc </FRAME> </VALENCY> <DOMAIN>
factotum </DOMAIN> </SYNSET>
   <SYNSET> <ID> ENG20-01918210-v </ID> <POS> v </POS>
<SYNONYM> <LITERAL> pozvednout <SENSE> 2 </SENSE>
</LITERAL> <LITERAL> dát nahoru <SENSE> 1 </SENSE>
</LITERAL> </SYNONYM> <ILR> ENG20-01916187-v <TYPE>
hypernym </TYPE> </ILR> <STAMP> Karel 2003/12/29
</STAMP> <VALENCY> <FRAME> pozvednout, dát nahoru
kdo1*AG(person:1)=co4*OBJ(object:1) %otec dal tu knihu
nahoru </FRAME> </VALENCY> <DOMAIN> factotum </DOMAIN>
</SYNSET>
```

The spaces between the individual xml tags and their values are not present in the original file. They have been added so that the formatting is nicer here.

3.5 WordNet versus ESP data

3.5.1 Introduction

Although the WordNet contains many semantic relations, the basic unit here is a synset. A synset is a group of expressions which are synonymous. So the smallest units in WordNet are in fact word meanings.

The task of this analyses is to compare label-sets from the ESP data against the WordNet synsets. The obvious question is how to compare apples and pears?

A synset contains synonyms and a label-set contains labels of which some might and some might not be semantically related. There are many types of semantical relations.

We have to bear in mind how every label-set originated. It is made up of labels describing a picture downloaded from the internet. A picture can contain anything and there are almost no limits, except for the length of the string, so there can be labels semantically related and labels without semantic relations. The labels with a semantic relation originate for instance when there are two objects in the picture, which have the semantical relation between them (i.e. 'door' and 'house'). If there are two object which are not related to each other then the labels can be semantically unrelated ('car' and 'dog'). But it gets tricky. There can be two unrelated objects (house and a car window lying on the street) in the picure and we still can get description *window* for the first object and *house* for the second, and the two labels are semantically related!

Between the labels of a house picture can be synonyms for house, i.e. *building*, *household*, *estate*.

In this analysis we decided to analyse synonymy because the WordNet is especially strong at synonyms. We are investigating synonymy also because if an object on a picture could be labeled with more generally used synonyms, it is almost sure, that it had been labeled with all of them.[4]

So we want to find out whether there are synonymous labels within ESP data label-sets and of course we want to know how many synonymous labels there were.

WordNet and its synsets are taken as a source of synonymous expressions in the analysis and we want to find out to what extend the ESP label sets are synonymous. `Synonymity` of a set of labels S is understood as number of different pairs of labels (lab_1, lab_2), such that lab_1 and lab_2 are syryonyms.

3.5.2 Computing the Synonymity of a Set of Labels

A set of labels does not necessarily have to be a label-set, it can be a `neighbor-set` (a set which contains neighbors from the ESP data), or any subset of the set of all ESP data labels.

For a set of labels we find how many of the possible pairs of labels are synonymous. We have the meter of being synonymous in the WordNet's synsets. So for each pair of labels we need to know if the two labels are synonymous or not.

It is easily done. If there is a synset which contains both the labels, we found a synonymous pair. We can count how many pairs of labels were synonymous for a given set of labels .

[4]Because the picture is no more being uploaded into new ESP game sessions only after a certain number of rounds passes during which no new labels are added. If the synonym is easy to think of, it is quite probable that some pair of players would bring it up.

3.5.3 Definition of method

Analysis 1

Analysis 1 is the simplest one. We compute how many labels in each label-set are synonymous. Then it can easily be computed how many per cents of a label-set are synonymous (i.e. they appear in a synset). When we compute this for all the label-sets, we compute and draw a histogram.

Analysis 2

The same as Analysis 1 with only one difference. Instead of taking the label-sets as in the first analysis, in the second one we use so called `filtered neighbor-sets`. A `neighbor-set` is a set that contains neighbors of some label.

The filtering depends on the threshold Z (see Chapter 1). A `filtered neighbor-set` originates from a neighbor-set NS when we keep only labels lab_1 for which: $\exists lab_2 \in NS, lab_2! = lab_1 : freq(lab_1, lab_2) \geq Z$.

A label $lab1$ is not filtered out of a neighbor-set only if the neighbor-set contains at least one more different label lab2, which has been together with lab1 at least Z-times.

So in the analysis 2 for each filtered neighbor-set we compute the percentage of synonymity in regard to the WordNet database. Because we have different filtered neighbor-sets for different thresholds, the analysis yields n different histograms for n different thresholds Z.

Analysis 3

The same as analysis 2, instead of taking filtered neighbor-sets, we take only filtered label-sets, which are much smaller units. Again here we have multiple histograms for the different thresholds.

Analysis 4

For each pair of neighbors with neighbor count $\geq Z$ we determine whether they are synonyms or not and then compute the precision and recall.

3.5.4 Results of the Czech WordNet versus Czech ESP Data

Histogram 1

The first analysis' histogram is saying that the majority (88.44%) of label-sets contain 0% to 5% of synonyms. Furthermore 99.99% of label-sets is 0% to 30% synonymous.

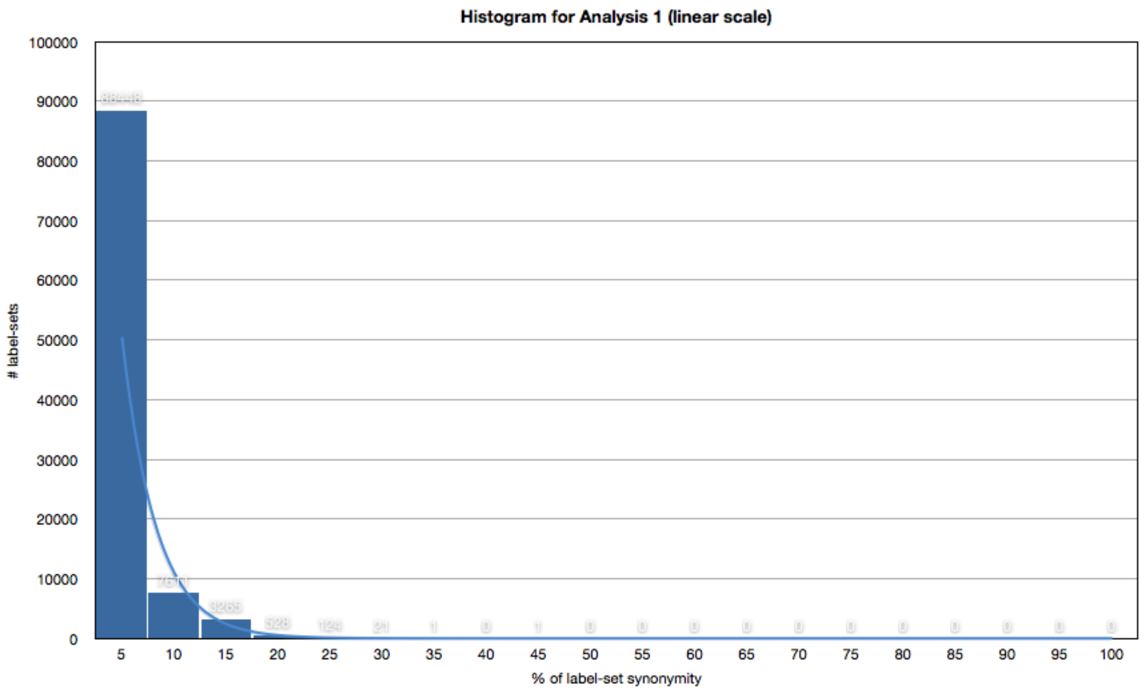


Figure 3.2: Histogram 1

Histogram 2

In the second and third analysis' histograms we had to use logaritmic scale because the range of values is very large. We used the filtered neighbor-sets here to determine whether the neighbor count metric would be useful when retrieving synonyms from the ESP data. If two labels occur together in n different label-sets, the probability that those two labels are synonymous could be higher.

However, the second analysis showed that neighbor-sets retrieved with bigger neighbor counts are not more synonymous. The trend is clearly readable from the histograms. The histograms are even sharper than the one in the analysis 1.

One minor bump against the trend lies within the 20%-25% interval of synonymity. There were 3 filtered neighbor-sets with Z = 50, but already 4 with Z = 100. It means this is the only place where the bigger neighbor count helped to discover more synonymous neighbor-sets, but it is such a small amount, that it is not significant at all.

Histogram 3

The idea with filtered neighbor-sets is wrong. The problem might be the fact that the neighbor-sets are too large. So in analysis 3 we focused back on label-sets

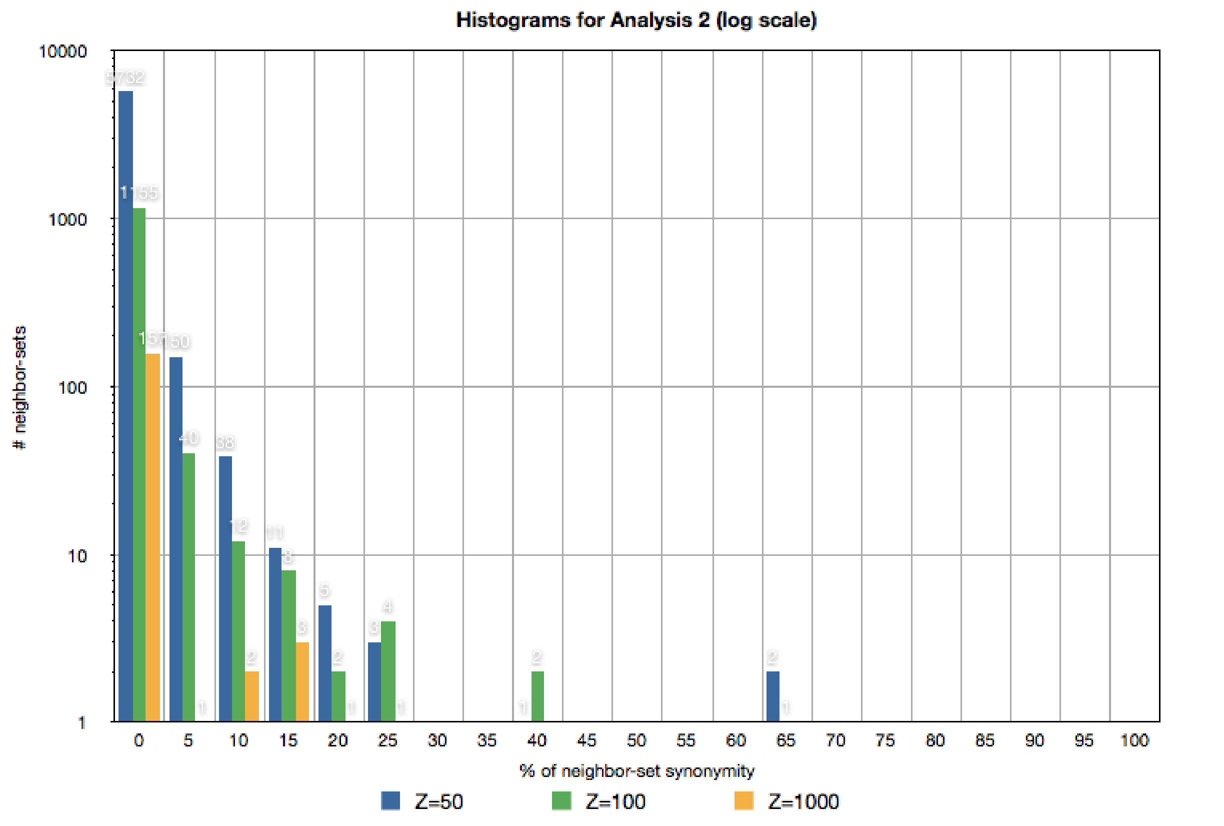


Figure 3.3: Histogram 2

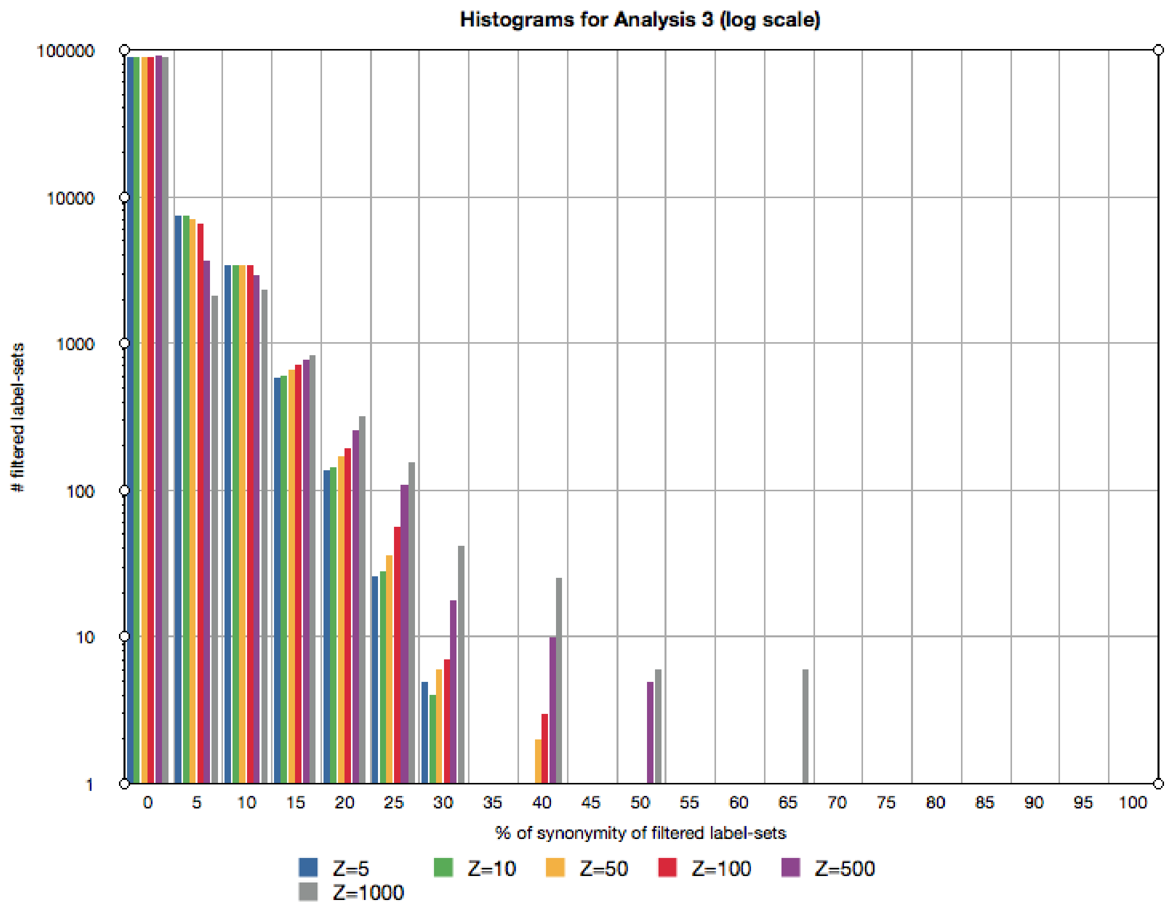


Figure 3.4: Histogram 3

and used the neighbor count based filtering on them. The surprise is it works a bit better. The general trend remains still the same with the majority of filtered label-sets (from 88.3% for Z = 5 to 93.9% for Z = 1000) being only 0 to 5% synonymous. However, there is significantly more discovered filtered label-sets from 15% to 70% of synonymity when using higher neighbor counts and here the trend is ”the more Z the more of synonymous label-sets.”

Results of the Analysis 4

Table 3.1 shows Precision and Recall computed in the Analysis 4 run on the Czech WordNet and ESP data.

Z	P(%)	R (%)
5	0.04	3,169.96
10	0.05	1,735.28
50	0.12	387.44
100	0.17	192.03
500	0.44	27.15
1,000	0.67	9.67

Table 3.1: Results of Analysis 4 (Czech)

Z	P(%)	R (%)
5	0.07	680.39
10	0.1	349.89
50	0.22	67.15
100	0.3	30.7
500	0.67	3.66
1,000	1.04	1.28

Table 3.2: Results of Analysis 4 (English)

3.5.5 Results of the English WordNet versus (English) ESP Data

Results of Analyses 1, 2, 3

The results of English Analysis 1, 2 and 3 can be seen in Figure 3.5, Figure 3.6 and Figure 3.7. The general trends resemble those from the Czech analysis, although at the same levels of synonymity, the English data show more sets of labels than the Czech data. For instance at 40%-45% of synonymity in the Analysis 3, there is only 25 filtered label-sets in the Czech ESP data, whereas there is 343 filtered label-sets in the original ESP data for the same $Z = 1000$.

Results of the Analysis 4

Table 3.2 shows Precision and Recall computed in the Analysis 4 run on the English WordNet and ESP data.

3.5.6 Conclusion

We could see that the ESP data failed on one of the most simple task of synonyms resolution. It is clear that the ESP data are rich and that they contain many synonyms, and other types of semantically related labels (*hand* - *arm*; *window* - *house*; *animal* - *dog*), which are contained in the WordNet database. The problem of our approach is that when we pick two labels out of an ESP data label-set, we

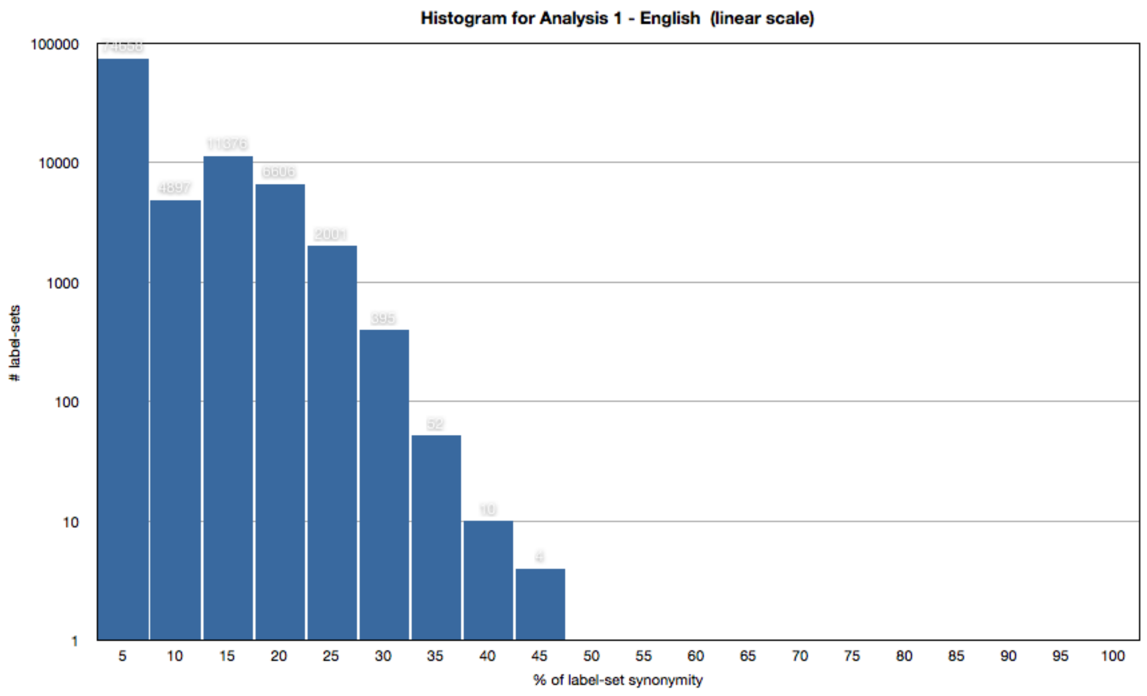


Figure 3.5: Histogram 1

have no idea whether they are semantically related at all and we have no means of algoritmically determining what type of semantical relation it is. The probabilistic approach definitely failed on synonyms.

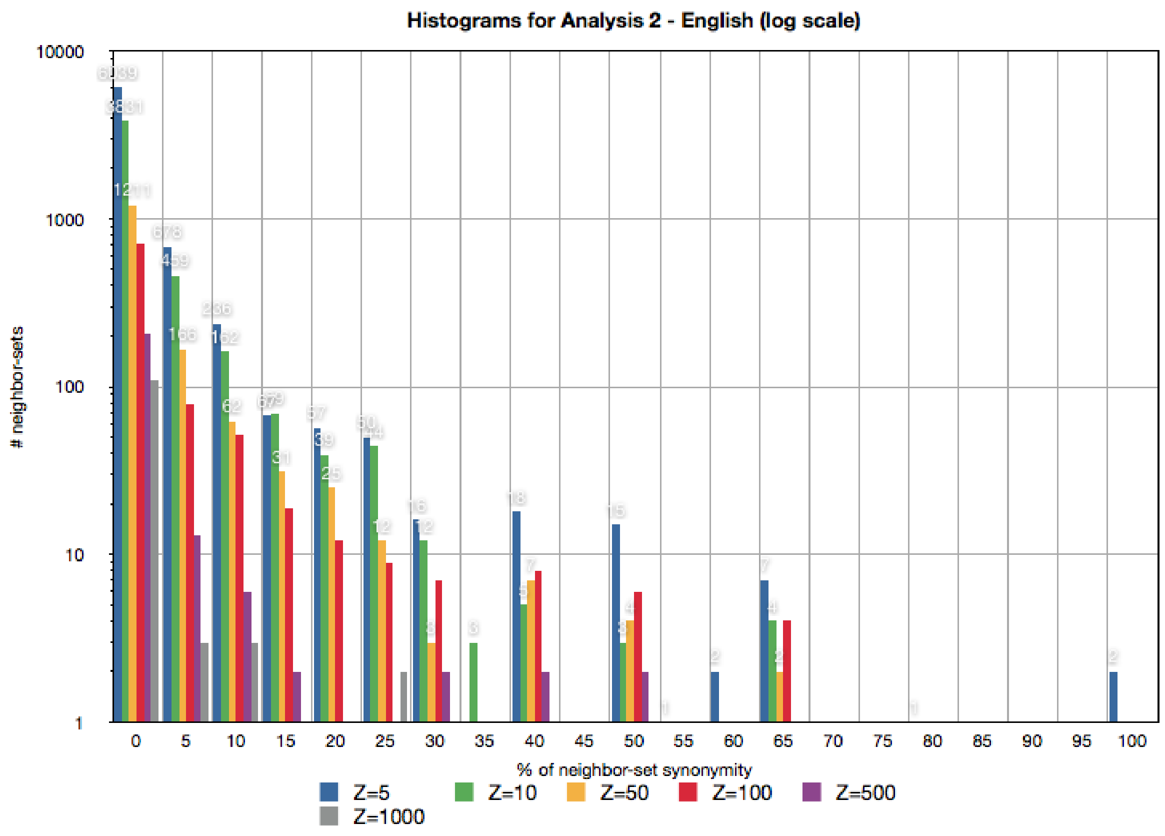


Figure 3.6: Histogram 2

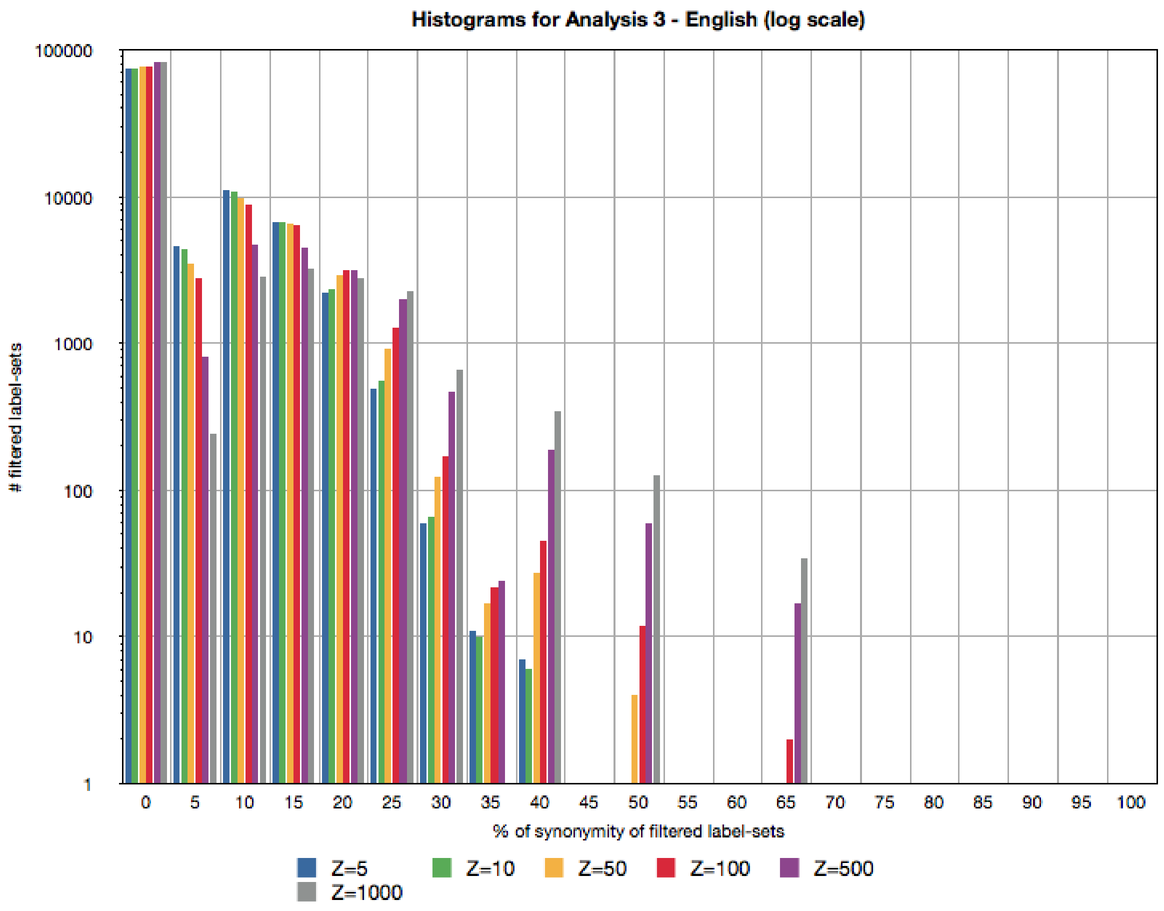


Figure 3.7: Histogram 3

Chapter 4

Conclusion

4.1 Benefits of the ESP Data to the WordNet Database

Although the large portions of the ESP database are informationally very rich, we did not find a method to employ the ESP data in finding coreferences or in searching for synonyms. Nevertheless, it might be of a help for the WordNet database. The neighborhood counts from a sample large enough could be taken as a measure of relatedness of synsets. This would bring some extra information which might be useful.

4.2 ESP Data Use for Automatic Coreference Resolution

The real benefit to the NLP task of automatic coreference resolution is quite small. The ESP game, for a given image, outputs its labels. The issue is that a label can be any word or expression anyhow envoked when seeing the picture. The labels usually describe the picture. However, given a pair of image and its labels produced by the ESP game, computers with no additional information have no sense of how the labels are related to each other. Human presented with the image and its labels could quite succesfully determine the relationships between the individual labels, but computers cannot and if they were able to see such relationships, the problem of finding coreference would have been solved by now.

We can see that generalizing here is too harsh for the language, too harsh for the problem of finding coreference. The expectation put on the statistical approach deployed in this bachelor thesis was simply too high. We cannot rely on computing coreferences in texts based on how often two labels appear together in the ESP data. Sometimes this approach finds a coreference, but because it does not take into account the semantics of the text, it is as the results have shown condemned to failure.

4.3 Possible Future Improvement and Alternatives

However, the ESP game is not the only game designed by Louis von Ahn. Another game from the same author, in fact inspired by the ESP game itself is `Peakaboom` (von Ahn, 2005). This human algorithm game has on the input the ESP game image label pairs. It outputs details about the position of the objects within the image itself. The database of the image labels pairs enhanced by this information would be more useful for the task of automatic coreference resolution presented in this thesis. Here we could recognize quite a few possible coreferences by finding for instance two labels refering to the same position within the image and by comparing the two positions within the image we could know the relation between the two labels (i.e. part of the whole, synonym, etc.).

So it could help finding specific kinds of coreferences. On the other hand it would be much more computationally demanding and the biggest problem would still not be solved: the actual semantics in the text. We might still for example correctly retrieve the information that window is part of a car, but not always when they appear in the text, must necessarily these two words be co-refering. There can be two cars mentioned in one paragraph of a text: a black car and a red car. The author writes about the red's car window being broken, so the pseudo-coreference window and car for the red car would be a correct coreference, but pseudo-coreference window and black car actually would not. And this is what our approach still could not solve.

User Documentation

Label Viewer

The `Label Viewer` is an application designed for user-friendly viewing the English or Czech labels and their relations. It has three main parts as depicted in Figure 4.1:

- **part 1 - list of labels**. Labels from the ESP data are displayed here. Each label has its `id` and `frequency`. Frequency means the number of label-sets containing the label. The label *cathedral* is selected in part 1 of the example.
- **part 2 - list of label-sets**. Selecting a label from part 1 reveals the list of id's of label-sets in which the label (i.e. *cathedral*) occurred (part 2 - left column). One can click on a label-set id to inspect the label-set's labels (part 2 - right column) and the original picture which belongs to the viewed label-set. In the Figure 4.1 the label-set with $id = 493$ is shown.
- **part 3 - list of neighbors**. In part 3, the neighbors for a selected label (i.e. *cathedral*) are shown in the `neighbors` column. The `frequency` here is the number of label-sets in which the selected label (*cathedral*) and the selected neighbor (*spire* in the right column in part 3) co-occurred together, i.e. $freq(w_1, w_2)$ described in 2.3.1. In our example 56 means that labels *cathedral* and *spire* co-occured together in 56 different label-sets.

Both lists of labels and neighbors can be resorted alphabetically or by frequency. By default, labels are sorted alphabetically and neighbors by frequency.

Label Viewer installation

It is intended for use on unix machines, with minimum 1024MB RAM and JVM 1.6 installed. When installing the program, first copy the whole directory `.../tools/LabelViewer/LABELVIEWER_v1.06` into the directory where you want to have it on your machine. Then change the working dictionary to `.../tools/LabelViewer/LABELVIEWER_v1.06/` located on your computer and run the instal script:

`./instal.sh`.

This script downloads the sample of 100,000 ESP images (size of 743MB) from the internet, which is the only thing that needs to be installed.

Running the Label Viewer

Change the current working dictionary to .../tools/LabelViewer/LABELVIEWER_v1.05/. Then start the program by running the script

`./run.sh [ -cz | -en ]`.

It takes around a minute and a half to first boot the program, during which time the relation of neighbors is determined and saved onto the disk for future use, so that next time the boot-up time of the program is even less. You are informed about the progress of loading the program and as soon as it is ready, the main view appears.

Chains Viewer

The `Chains Viewer` program is an application designed for computing the pseudo-coreference chains over a sample Czech or English text in the csts[1] format based on the ESP data. It compares the pseudo-coreference chains against the manually annotated coreference chains and lets user browse through the pseudo-coreference and coreferential chains. The screenshot of the program is in Figure 4.2.

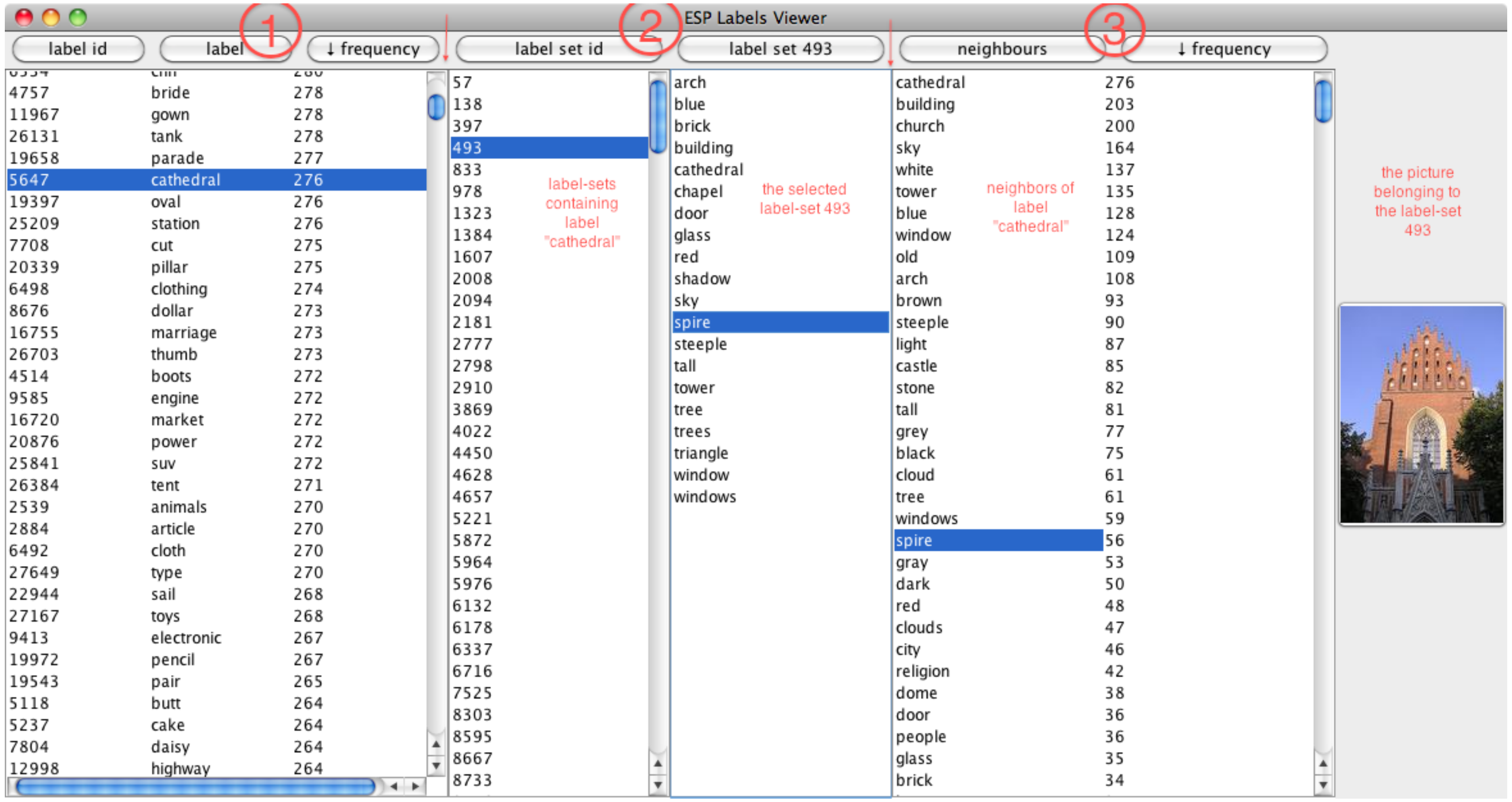
The program consists of two parts. The controls, which allow the user to browse through the coreferences and pseudo-coreferences and the display, which shows the coreferences and pseudo-coreferences within the text. The controls are located on the left side of the program and the display is located in the central part of the program.

The controls and display are further divided into two parts. The upper part deals with coreference chains and the lower part with pseudo-coreference chains.

Chains Viewer installation

It is intended for use on unix machines, with minimum 1024MB RAM and JVM 1.6 installed. The program itself does not have to be installed, you can copy the directory ChainsViewer anywhere on the disk.

[1]Czech Sentence Tree Structure is a format developed at ÚFAL, Charles University in Prague for Czech National Corpus and Prague Dependancy Treebank. `https://wiki.ufal.ms.mff.cuni.cz/format-csts`


ESP Labels Viewer
1
2
3
label-sets containing label "cathedral"
the selected label-set 493
neighbors of label "cathedral"
the picture belonging to the label-set 493

label id	label	↓ frequency
4757	bride	278
11967	gown	278
26131	tank	278
19658	parade	277
5647	cathedral	276
19397	oval	276
25209	station	276
7708	cut	275
20339	pillar	275
6498	clothing	274
8676	dollar	273
16755	marriage	273
26703	thumb	273
4514	boots	272
9585	engine	272
16720	market	272
20876	power	272
25841	suv	272
26384	tent	271
2539	animals	270
2884	article	270
6492	cloth	270
27649	type	270
22944	sail	268
27167	toys	268
9413	electronic	267
19972	pencil	267
19543	pair	265
5118	butt	264
5237	cake	264
7804	daisy	264
12998	highway	264

label set id
57
138
397
493
833
978
1323
1384
1607
2008
2094
2181
2777
2798
2910
3869
4022
4450
4628
4657
5221
5872
5964
5976
6132
6178
6337
6716
7525
8303
8595
8667
8733

label set 493
arch
blue
brick
building
cathedral
chapel
door
glass
red
shadow
sky
spire
steeple
tall
tower
tree
trees
triangle
window
windows

neighbours	↓ frequency
cathedral	276
building	203
church	200
sky	164
white	137
tower	135
blue	128
window	124
old	109
arch	108
brown	93
steeple	90
light	87
castle	85
stone	82
tall	81
grey	77
black	75
cloud	61
tree	61
windows	59
spire	56
gray	53
dark	50
red	48
clouds	47
city	46
religion	42
dome	38
door	36
people	36
glass	35
brick	34

Figure 4.1: The Screenshot of the Label Viewer Program

Running the Chains Viewer

Run the program by changing the current working dictionary to .../tools/ChainsViewer/CHAINS_vX.X and then execute the shellscript which is located within the ”CHAINS_vX.X” directory by typing

`./run.sh [ -cz | -en ].`

The program loads the ESP data, loads the input csts file which contains the text to process (A.C.Doyle), performs the search for pseudo-coreference chains and compares the annotated coreference chains with the pseudo-coreference chains found by the algorithm.

It takes the program around two minutes to boot on a 2.26GHz dual core processor with 4GB of RAM.

When the program is ready, the controls of the program are enabled and you are able to browse through the text and the coreference and pseudo-coreference chains as described below.

The displays

As soon as the program is loaded and ready, which is when the control buttons are no more grayish, both the displays show the input text. The words with the gray background are the words that are part of a coreference chain (in the upper display) or a pseudo-coreference chain (in the lower display). No coreferences nor pseudo-coreferences are shown yet.

The upper display displays the coreferences. From the definition, all the words that are corefering to the same word are part of one big coreference. So if a word is part of a coreference chain, it is the only chain it is part to.

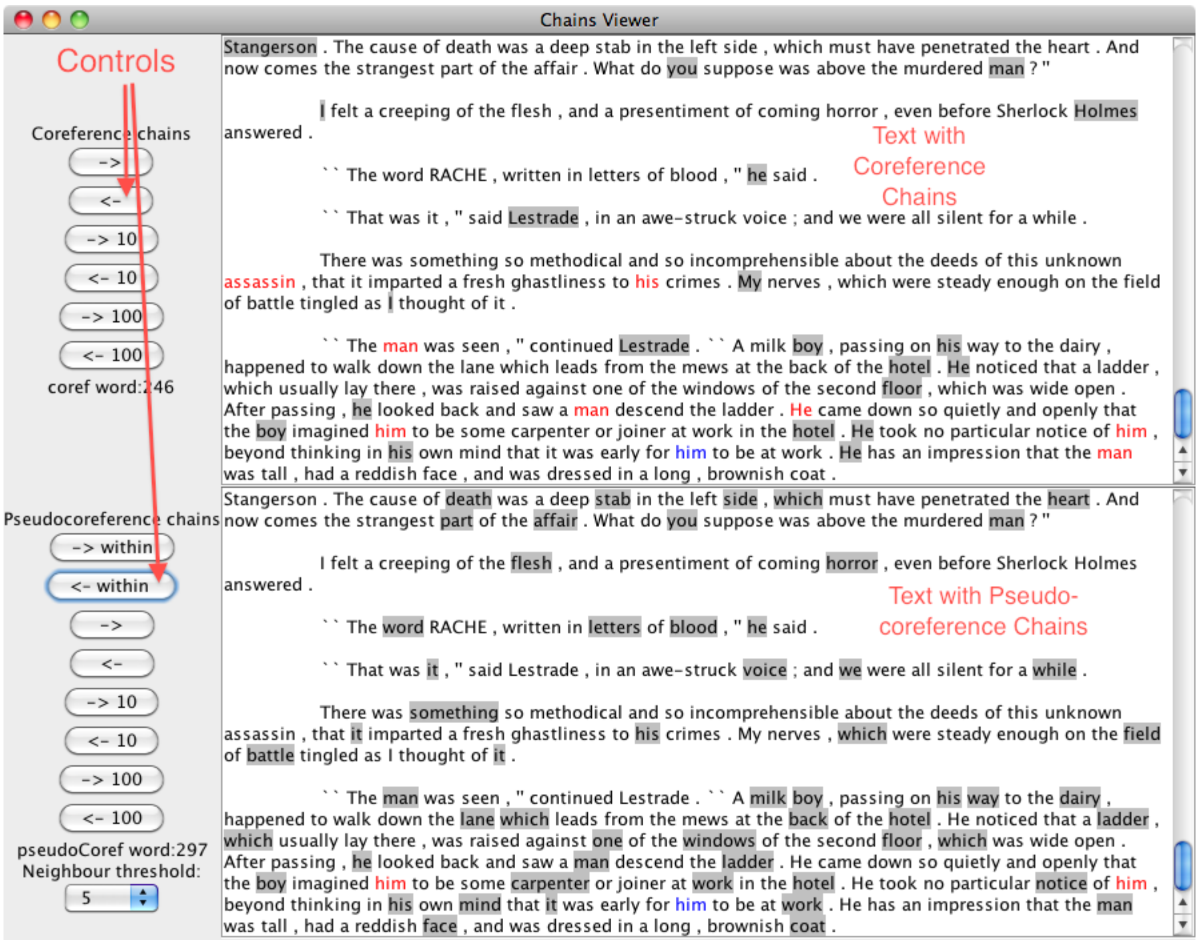
The lower display displays the pseudo-coreferences. Here a word, if part of a pseudo-coreference, it can be at the same time part of more different pseudo-coreferences.

Browsing through the coreference and pseudo-coreference chains in the input text

Coreference chains

To browse through the coreferences click the button "$\rightarrow$" or "$\leftarrow$". This moves the current word, which is highlighted by blue color, back and forth between words that are part of some coreference. The blue word determines which coreference is shown. It is the one that contains the blue word. The other words that are part of the currently selected coreference are displayed in red.

You can move the blue word not only to the nearest possible word, i.e. by 1, but also by 10 and 100 via the appropriate buttons "$\rightarrow$ 10" and "$\leftarrow$ 10", resp. "$\rightarrow$ 100"



Coreference chains

coref word:246

Pseudocoreference chains

pseudoCoref word:297

Neighbour threshold: 5

Stangerson . The cause of death was a deep stab in the left side , which must have penetrated the heart . And now comes the strangest part of the affair . What do you suppose was above the murdered man ? "

I felt a creeping of the flesh , and a presentiment of coming horror , even before Sherlock Holmes answered .

`` The word RACHE , written in letters of blood , " he said .

`` That was it , " said Lestrade , in an awe-struck voice ; and we were all silent for a while .

There was something so methodical and so incomprehensible about the deeds of this unknown assassin , that it imparted a fresh ghastliness to his crimes . My nerves , which were steady enough on the field of battle tingled as I thought of it .

`` The man was seen , " continued Lestrade . `` A milk boy , passing on his way to the dairy , happened to walk down the lane which leads from the mews at the back of the hotel . He noticed that a ladder , which usually lay there , was raised against one of the windows of the second floor , which was wide open . After passing , he looked back and saw a man descend the ladder . He came down so quietly and openly that the boy imagined him to be some carpenter or joiner at work in the hotel . He took no particular notice of him , beyond thinking in his own mind that it was early for him to be at work . He has an impression that the man was tall , had a reddish face , and was dressed in a long , brownish coat .

Stangerson . The cause of death was a deep stab in the left side , which must have penetrated the heart . And now comes the strangest part of the affair . What do you suppose was above the murdered man ? "

I felt a creeping of the flesh , and a presentiment of coming horror , even before Sherlock Holmes answered .

`` The word RACHE , written in letters of blood , " he said .

`` That was it , " said Lestrade , in an awe-struck voice ; and we were all silent for a while .

There was something so methodical and so incomprehensible about the deeds of this unknown assassin , that it imparted a fresh ghastliness to his crimes . My nerves , which were steady enough on the field of battle tingled as I thought of it .

`` The man was seen , " continued Lestrade . `` A milk boy , passing on his way to the dairy , happened to walk down the lane which leads from the mews at the back of the hotel . He noticed that a ladder , which usually lay there , was raised against one of the windows of the second floor , which was wide open . After passing , he looked back and saw a man descend the ladder . He came down so quietly and openly that the boy imagined him to be some carpenter or joiner at work in the hotel . He took no particular notice of him , beyond thinking in his own mind that it was early for him to be at work . He has an impression that the man was tall , had a reddish face , and was dressed in a long , brownish coat .

Figure 4.2: The screenshot of the Chains Viewer program

and "$\leftarrow$ 100".

Pseudo-coreference chains

The principle is the same, with one exception: a word here can be member of more than one pseudo-coreference. So when moving the current word, which is selected by blue, there may be more pseudo-coreferences related to this word. Thus only the first pseudo-coreference is displayed in red letters. For accessing the other pseudo-coreferences related to the current word, the buttons "$\rightarrow$ within" and "$\leftarrow$ within" are reserved. By those buttons you can iterate within all pseudo-coreferences that are related to the blue word.

When viewing pseudo-coreferences user can choose (in the lower left corner) between different `Neighborhood thresholds` which have impact on finding the pseudo-coreference chains.

Program Documentation

Label Viewer

Description of the structure of the whole program, of classes, methods, etc. This part of documentation is mainly contained also in the javadoc documentation genrated directly from the source codes into html format. There is only a basic description here.

Structure of the program

The program is part of the package dictionary. The package dictionary is designed for working with the ESP data in specified format and displaying it in a user-friendly manner through graphical user interface.

Class GUI is the main class. It creates a new instance of Data and when the data are loaded, the class GUI displays them in the graphical user interface. This class is designed for creating and maintaining the GUI for viewing the ESP data.

Class Data reads ESP data from a file and then stores this data in the program memory.

LabelSet LabelSet represents a label-set from the ESP data.

Description of important classes, interfaces, etc.

Class Data

This class reads ESP data from a file and then stores this data in the program memory. It includes methods for accessing the data.

Field Detail

- `ready`

  public static volatile boolean ready

  As soon as the initialization phase of the program has ended, ready is set to true.

Constructor Detail

- `public Data(String lang)` throws IOException, FileNotFoundException

  Constructor of the class Data initializes class members, reads the ESP data from 2 files and stores it into program data structures for future usage.

Method Detail

- `public Integer getIndex(String label)`

  If the label is present, returns its id, else returns null.

- `public String getLabel(int id)`

  Returns the label with the specified id.

  Parameters: id - the label id.

- `public HashMap getNeighbours(int label_id)`

  Returns the neighbors of the label as a HashMap

- `public String returnFileName(int j) throws IOException`

  Returns the original file in which the label-set j was saved.

- `protected HashMap<Integer,Integer> findNeighbours4(int i, boolean sortAbc)`

  Return the neighbors of the label.

  Parameters: i - the index for retrieving the label id; sortAbc - true -> labels sorted alphabetically, false-> by frequency

- `protected HashMap<Integer,Integer> findNeighbours3(int i, boolean sortAbc)`

  Counts the neighbors and neighborhood frequencies for the label i.

- `public List<Integer> getLabelSetIds(int labelId)`

  Returns the list of id's of label-sets in which the label with labelId occurred.

  Parameters: labelId - the id of the label.

- `public LabelSet getLabelSet(int id)`

  Get the label-set with this id.

  Parameters: id - the label-set id.

Class GUI

This class is designed for creating and maintaining the GUI for viewing the ESP data. It is the main class, it contains the method main, which launches the whole program.

Constructor Detail

- `public GUI()`

Method Detail

- `public static void createAndShowGUI(String paramLang) throws IOException`

  This method creates the starting progressBar, runs the ESP data initialization in background and after it is done, switches to the main GUI.

- `public static void main(String[] args)`

  It initializes the program and starts the GUI.

  Parameters: args - no arguments expected

Class LabelSet

LabelSet represents a label-set from the "ESP data". It contains different labels and provides methods for working with the label set. To represent label not the string representation, but the label index is used instead.

Field Detail

- `protected List<Integer> set`

  label indexes.

- `protected List<Integer> counts`

  how many times a label occured in the ESP data.

Constructor Detail

- `public LabelSet()`

  Create an empty LabelSet.

Method Detail

- `public void addLabel(Integer hash)`

  Adds a label index into the LabelSet.

- `public void addLabel(Integer id, Integer count)`

  Adds a label index with specified count into the LabelSet.

  Parameters: id - the label id; count - the count

- `public void setFileName(String fileName)`

  Sets the file name of the LabelSet.

  Parameters: fileName - the file name to be set; getFileName

- `public String getFileName()`

  Gets the file name of the LabelSet. Returns the file name as a String.

- `public void delete()`

  Deletes all the labels from the label set.

- `public int size()`

  Returns the size of the LabelSet (how many labels are contained in the LabelSet).

- `public int get(int index)`

  Returns the label id saved at the index.

- `public int getCount(int index)`

  Returns the label count saved at the index.

- `public void print()`

  Prints the label indexes from the LabelSet

- `public void printAll()`

  Prints the label indexes and the label counts from the LabelSet

Chains Viewer

Description of the structure of the whole program, of classes, methods, etc. This part of documentation is mainly contained also in the javadoc documentation genrated directly from the source codes into html format. There is only a basic description here.

Structure of the program

The package chains_console is a program designed to read a "text" (in xml or txt format) with manually annotated coreferences and to compute the coreference chains and pseudocoreference chains based on the ESP data. It also performs analysis of coreference vs. pseudo-coreference chains.

Description of important classes, interfaces, etc.

Class Chain

This class represents a chain (a set of words).

Constructor Detail

- `public Chain()`

  The constructor of the class.

Method Detail

- `public Chain makeCopy()`

  Creates a deep copy of this chain.

  Returns: the deep copy of this chain.

- `public void addWord(Word w)`

  Adds a word to this chain.

  Parameters: w - the word that is added to this chain.

- `public java.lang.String toString()`

  Overrides: toString in class java.lang.Object

  Returns: string in this format: #words[Coordinate1]'word_1', [Coordinate2]'word_2', ..., [Coordinate_n]'word_n'.

- `public boolean equals(java.lang.Object o)`

  Overrides: equals in class java.lang.Object

- `public int hashCode()`

  Overrides: hashCode in class java.lang.Object

Class Coordinate

This class represents the coordinate of a word in the text (paragraph_index + word_index).

Constructor Detail

- `public Coordinate(int para,int word)`

  The constructor of the class Coordinate.

  Parameters: para - the paragraph index; word - the word index

Method Detail

- `public java.lang.String toString()`

  Overrides: toString in class java.lang.Object

  Returns: string in this format: "[paragraph_index,word_index]".

- `public boolean equals(java.lang.Object o)`

  Two Coordinates are equal if their paragraph indexes are the same, and their word indexes are the same.

  Overrides: equals in class java.lang.Object

  Returns: true, if this Coordinate is equal to o, otherwise false

- `public int hashCode()`

  Overrides: hashCode in class java.lang.Object

Class CorefListener

It is listening on the coreference controls in the GUI and if an event arises, it correctly serves it.

Constructor Detail

- `public CorefListener(int delta)`

Method Detail

- `public void actionPerformed(java.awt.event.ActionEvent ae)`

  Specified by: actionPerformed in interface java.awt.event.ActionListener

Graph

The class Graph finds the coreference chains (method startAllDfs) and finds the pseudcoreferential chains (method startAllDfs1).

Method Detail

- `public java.lang.String toString()`

  Overrides: toString in class java.lang.Object

- `protected void startDfs1(int i)`

  Start DFS to find pseudocoreferential chains from the i-th vertex.

  Parameters: i - the i-th vertex

Main

The main class.

Constructor Detail

- `public Main()`

Method Detail

- `public static void main(java.lang.String[] args)`

  The main method.

  Parameters: args - a fileName to read the book from (in xml format), and an optional -compare parameter (can be used if the xml file already contains coref tags.)

PseudoCorefListener

It is listening on the pseudo-coreference controls in the GUI and if an event arises, it correctly serves it.

Constructor Detail

- `public PseudoCorefListener(int delta)`

Method Detail

- `public void actionPerformed(java.awt.event.ActionEvent ae)`

  Specified by: actionPerformed in interface java.awt.event.ActionListener

Word

The class Word represents the word in the text and items related to it.

Field Detail

- protected HashSet<Coordinate> backCoords

Constructor Detail

- `public Word(java.lang.String word, boolean nospace, Coordinate c)`

  The constructor of the class Word.

  Parameters: word - the word itself; nospace - whether or not there is no space after the word; c - the Coordinate of the word.

Method Detail

- `public java.lang.String toString()`

  Overrides: toString in class java.lang.Object

  Returns: the string representation

- `public Coordinate getCoordinate()`

  Gets the coordinate of this word.

  Returns: the word's Coordinate

- `public java.lang.String getWord()`

  Gets the word itself.

  Returns: the word itself as a string

- `public void addRef(int r)`

  Adds a reference r to the word.

  Parameters: r - the reference = label_id; startIndex

- `public void startIndex(int start)`

  Assigns the start index to the word.

  Parameters: start - the start index; paintWord

- `public void paintWord(int colour, javax.swing.text.StyledDocument doc)`

  Paint the word with the color. The word color is assigned, and the word in the StyledDocument gets corresponding text color.

  Parameters: colour - the color; doc - the StyledDocument containing the text; paintWord

- `public void paintWord(int colour,Coordinate c)`

  Paints the word with the colour.

  Parameters: colour - the color to paint; c - the Coordinate of the word

- `public void unpaint(javax.swing.text.StyledDocument doc)`

  Un-paints the word. The style is changed to black ink.

  Parameters: doc - the StyledDocument containing the text (either doc or doc2); isPainted

- `public boolean isPainted()`

  Returns: true if the word is painted, otherwise false.

- `public int refSize()`

  Returns: the #label-references

- `public void addPOS(java.lang.String tag, boolean POS)`

  Adds a POS tag to the word.

  Parameters: tag - the POS tag; POS - whether the POS tag is sure(true) or only probable (false)

- `public java.lang.String getPos()`

  Gets the POS tag.

  Returns: the POS tag

- `public boolean isNoSpace()`

  Returns: true if no space should be put after this word, otherwise false.

- `public void rewriteWord(java.lang.String newWord, boolean noSpace)`

  Rewrites this word with a new one.

  Parameters: newWord - the new word, which replaces the old one; noSpace - even the nospace variable has to be rewritten; isBadlyTagged

- `public boolean isBadlyTagged()`

  Some POS tag are only "probable", the word is not matching the original one.

  Returns: true if this is the case, otherwise false

- `public void addColour(int colour)`

  Adds a colour to the list of colours.

  Parameters: colour - the colour to add

- `public void addBackCoord(Coordinate c)`

  Adds a Coordinate to the backCoords.

  Parameters: c - the Coordinate to add

- `public void deleteCoref()`

- `public void setGraphNode(List<Word> children)`

- `public boolean equals(java.lang.Object o)`

  Two words are equal if their Coordinates are equal.

  Overrides: equals in class java.lang.Object

  Returns: true if this word is equal to o, otherwise false.

- `public int hashCode()`

  Overrides: hashCode in class java.lang.Object

References

Luis von Ahn. 2005. *Human Computation.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA.

Luis von Ahn and Laura Dabish. 2004. Labeling Images with a Computer Game. In Proceedings of *Chi 2004*, pp. 24–29, Vienna, Austria.

Arthur Conan Doyle. 1887, 2005. *A Study in Scarlet.* `http://www.gutenberg.org/ebooks/244.`

Donald E. Knuth 1997. *The Art Of Computer Programming Vol 1. 3rd ed.* Boston: Addison-Wesley.

John Hopcroft and Robert Tarjan. 1973. Efficient algorithms for graph manipulation. In *Communications of the ACM 16*, pp. 372–378.

Kristina Toutanova, Dan Klein, Christopher Manning and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proceedings of HLT-NAACL*, pp. 252–259.

List of Tables

2.1 The ESPgame100k data: statistics 12
2.2 Example of the dictionary format 13
2.3 Looking up the label *twelve* in the dictionary 14
2.4 The excerpt from the dictionary used for translating the label-set. Shortened. 15
2.5 Neighbor frequencies in a label-set. 18
2.6 Neighborhood graph of a label-set given by matrix. 18
2.7 Pseudo-coreference chains in *A Study in Scarlet* - no stems used . . 22
2.8 Pseudo-coreference chains in *Studie v Šarlatové* - no stems used . . 22

3.1 Results of Analysis 4 (Czech) 34
3.2 Results of Analysis 4 (English) 34