Parallel word sense disambigation of Hyperlex Algorithm

Parallel word sense disambigation of Hyperlex Algorithm
Zur Erlangung des Grades eines Doktor der Philosophie (Dr. phil.)
genehmigte Dissertation von Viswanath Vadhri aus Darmstadt
Tag der Einreichung: December 27, 2016, Tag der Prüfung: December 27, 2016
Darmstadt — D 17

1. Gutachten: Prof. Dr. Felix Wolf
2. Gutachten: Sebastian Rinke

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Laboratory for Parallel Programming

Parallel word sense disambigation of Hyperlex Algorithm
Parallel word sense disambigation of Hyperlex Algorithm

Genehmigte Dissertation von Viswanath Vadhri aus Darmstadt

1. Gutachten: Prof. Dr. Felix Wolf
2. Gutachten: Sebastian Rinke

Tag der Einreichung: December 27, 2016
Tag der Prüfung: December 27, 2016

Darmstadt — D 17

Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den December 27, 2016

_____

(V. Vadhri)

Abstract

Word sense disambiguation (WSD) is the ability to identify the intended meanings of words (word senses) in a context. It is a central research topic in Natural Language Processing (NLP). Word sense disambiguation is often characterized as an intermediate task, which is not an end in itself, but essential for many applications requiring broad-coverage language understanding. Examples include machine translation, information retrieval, information extraction or text mining.

Hyperlex is an unsupervised graph-based technique used in Natural Language Processing that is capable of automatically determining word senses from a large text base without recourse to a dictionary with excellent precision. In this thesis, we design and implement the Hyperlex algorithm as a scalable application which runs on the cluster and can process data of magnitude Terabytes. We analyse the results of the implementation.

Contents

1 Introduction

Word sense disambiguation is about automatically recognizing which of multiple meanings of ambiguous words is being used in a specific sentence. Words have different properties one of them is polysemy. Polysemy means many words have multiple senses. Here is an example 'Lets have a drink in the bar'. The word bar in this sentence means a drinking establishment, however in general the word bar has many other meanings. so for example 'bring me a chocalate bar'. 'I have to study for the bar', the meaning of bar in this case it means some exam for lawyers.

Lets have a drink in the bar Bring me a chocolate bar. I have to study for the bar.

Here are a few meanings taken from the wordnet dictionary for the word 'bar'

1. barroom, bar, saloon, ginmill, taproom - a room or establishment where alcoholic drinks are served over a counter; "he drowned his sorrows in whiskey at the bar"

2. bar - a counter where you can obtain food or drink; "he bought a hot dog and a coke at the bar"

3. prevention, bar - the act of preventing; "there was no bar against leaving"; "money was allocated to study the cause and prevention of influenza"

4. legal profession, bar, legal community - the body of individuals qualified to practice law in a particular jurisdiction; "he was admitted to the bar in New Jersey"

So the main goal of WSD here is to take a word like bar that has multiple senses and then in a given sentence findout which of those meanings is being used. So we need have the entire context, so we typically get access to the entire sentence and often to the entire paragraph in which the word appears. [1]

2 Application

Word sense disambiguation is very important component of Natural Language processing systems because its used for example to come up with for example semantic represntations of sentences and also for question-answering and for things like maching translation. Obviously if a word is ambiguous in one language it doesnt necessarily mean it is also ambiguous in another language. So for example if we want to translate the word 'play' from English to Spanish. We need to understand what is the object being played, so for example in english we say 'play the violin', in spanish this is translated with the verb 'tocar el violin' and then if the sentense is 'play tennis' we are going to use a different verb 'jugar al tennis'. So 'tocar' and 'jugar' are two different translations of the word 'play' and if we couldnt do proper word sense disambiguation in English we wouldnt be able to translate those sentences into spanish properly.

Other uses of WSD include Accent restoration, for example the word 'cote' in frence means different things when its pronounced differently depending upon where the accents appear Text-to-speech generation, In this case we may have a word that has multiple pronunciations depending on the meaning for example 'lead'(action) or 'lead'(metal) Spelling correction, we want to be able to distinguish between words like 'aid' and 'aide' Capitalization restoration, If we have a word like 'turkey', if we know its meaning as a bird or a country we can properly capitalize in a text in which it is not capitalized.

[1] Deutschland hat (noch) keine Amtssprache.

3 WSD: State of the Art

3.1 Lesk Algorithm

some of the common techniques used in word sense disambiguation starting a very old method by 'Michael Lesk'. It is called dictionary method. The idea behind this method is that if you have ambiguous words in a sentence that appear together, you are going to find all the possible meanings of each of those words and then look up for all the dictionary definitions of all the sense of those words and then look for pairs of dictionary definitions, one for each of the ambiguous words that overlap the most.

For example, lets look at the sentence that has two ambiguous words. 'The leaf is the food making factory for the green plants'. The ambiguous words in this sentence are 'plant' and 'leaf'. These words have many diffferent senses listed in the dictionary, lets look at two. Definitions of plant: plant1 : a living thing that grows in the ground, usually has leaves or flowers, and needs sun and water to survive. plant2 : a building or factory where something is made. Definintions of leaf: leaf1 : a lateral outgrowth from the plant stem that is typically a flattened expanded variably shaped greenish organ, function primarily in food manufacture by photosynthesis leaf2 : a part of the book or a folded sheet that contains a page on each side

So we have the word leaf, the word plant and we want to determine which of those sense is used in the sentence. So we have four possible combinations, leaf1plant1, leaf2plant2, leaf1plant2, leaf2plant1. We are going to look for the overlap in the dictionary definitions and pick the one that has the largest overlapping.

4 Decision Lists

5 WSD Approaches

There are majorly two approaches for Word Sense Disambiguation, which are Supervised learning approach and Unsupervised learning approach. Knowledge based approach is also another approach which has been implemented by few older systems. Below are the description given for these approaches.

5.1 Knowledge Based Approach

Knowledge based approach for WSD involves use of dictionaries, thesaurus, ontologism, etc. to understand the sense of words in context. Even though these methods have comparatively lower performance than other approaches, but one advantage of this approach is that they do have large-scale knowledge resources.

5.2 Supervised Learning Approach

Supervised learning method is that method which tries to find relationships between independent variables also known as input attributes and a target attribute also known as dependent variable. It makes use of labeled training data to derive functions. These derived functions are used further to predict results.

In supervised approaches, use of training data is involved. Generating training data manually requires lot of manual efforts plus the data canâĂŹt be trusted on its accuracy. Since the training data does not have the inputs classified correctly, this can result in getting wrong disambiguated results

5.3 Unsupervised Learning Approach

In Unsupervised learning method it tries to find hidden structure in unlabeled data. Unsupervised methods for WSD can be broadly divided into two categories namely vector clustering-based and graph based ones. For graph based methods generally unsupervised approach is preferred since it offers an advantage of not requiring the training data. Clustering âĂŞ finding natural groupings of items. According to Markov clustering algorithm, considering a graph, there will be many links within a cluster, and fewer links between clusters. This means if you were to start at a node, and then randomly travel to a connected node, youâĂŹre more likely to stay within a cluster than travel between clusters.

6 Hyperlex

Hyperlex is an excellent graph based unsupervised approach that is capable of automatically determining word uses in a textbase without recourse to a dictionary. The algorithm makes use of the specific properties of word co-occurance graphs, which are shown as having small-world properties. The small-world nature of a graph can be explained in terms of its clustering coefficient and characteristic path length. The clustering coefficient of a graph shows the extent to which nodes tend to form connected groups that have many edges connecting each other in the group, and few edges leading out of the group. On the other side, the characteristic path length represents âĂIJclosenessâĂİ in a graph. Randomly built graphs exhibit low clustering coefficients and are believed to represent something very close to the minimal possible average path length, at least in expectation. Perfectly ordered graphs, on the other side, show high clustering coefficients but also high average path length. According to Watts and Strogatz (1998), small-world graphs lie between these two extremes: they exhibit high clustering coefficients, but short average path lengths.

HyperLex algorithm accepts a collection of sentences including the target word as an input. Then it induces the senses of the target words as follows: (1) create a co-occurrence graph whose nodes are words in the context of the target word, (2) the nodes which represents the sense, called âĂŸhubâĂŹ, are identified, (3) the graph is subdivided into several sub-trees whose root nodes are the identified hubs, (4) and finally a sense of a word in a new context is disambiguated using the sub-trees. The detail of each step will be described in the following sections.

6.1 Building cooccurance graphs

Hyperlex depends on two most important charecterestics of to extract sense from words. They are frequency and Co-occurance. Frequency is the number of times a word occurs in the text corpora. A target word is said to cooccur with another word if they both appear in the same context. The cooccurance count is the number of times the two words cooccur in the corpora. These two parameters form the basis of calculating weights of the edges of the graph.

Only nouns and adjectives are considered to be the target words for disambiguation as including verbs causes a notable decline in performance since too many verbs like *can* and *start* have very general uses and they tend to cooccur with lot of words and do not help in the disambiguation process. Words with less than 10 occurances in the entire subcorpus should be discarded. Contexts containing fewer than 4 words after filtering should be deleted and only those cooccurances with a cooccurance count of 5 or should be retained.

A graph G(V,E) is constructed for every target word that is to be disambiguated with all the cooccuring words as the vertices V. The edges of the graph depend on whether the cooccuring words of the target word are cooccuring themselves.

The edges of the graph are assigned weights which depends on the frequency of the connecting words and the cooccurance count. The weight decreases as the cooccurance count increases. It is calculated as

$$W_{A,B} = 1 - max[p(A|B), p(B|A)]$$

where $p(A|B)$ denotes the conditional probability of observing A in a given context, knowing that context contains B, and inversely, $p(B|A)$ is the probability of observing B in a given context, knowing that it contains A. The probabilities depend on the frequencies of the individual words and the cooccurance count.

$$p(A|B) = f_{A,B}/f_B,$$

$$p(B|A) = f_{B,A}/f_A$$

Veronis implemented Hyperlex on ten highy polysemous words. Table 2 shows the number of contexts in which the word pairs *eau - ouvrage* (water - work) and *eau - potable*(water - drinkable) appear together or seperately in the *barrage* subcorpus.

$p(eau|ouvrage) = 183/479 = 0.38$, $p(ouvrage|eau) = 183/1057 = 0.17$, $w = 1 - 0.38 = 0.62$.

$p(eau|potable) = 63/63 = 1$, $p(potable|eau) = 63/1057 = 0.06$, $w = 1 - 1 = 0$.

The weight of the edge reflects the magnitude of the semantic distance between words. When the words always cooccur then the weight is equal to 0 and when the words never cooccur then the weight is 1. Edges with weight above 0.9 should be eliminated to make sure that only strong associations are included in the graph. Otherwise the graph tends to become totally connected as the corpus grows in size, due to accidental cooccurances of any word pairs.

Table 1: Number of cooccurances of eau-ouvrage(water-work) and eau-potable(water-drinkable)

	EAU	∼EAU	Total
OUVRAGE	183	296	489
∼OUVRAGE	874	5556	6430
Total	1057	5822	6909
POTABLE	63	0	63
∼POTABLE	994	5852	6846
Total	1057	5852	6909

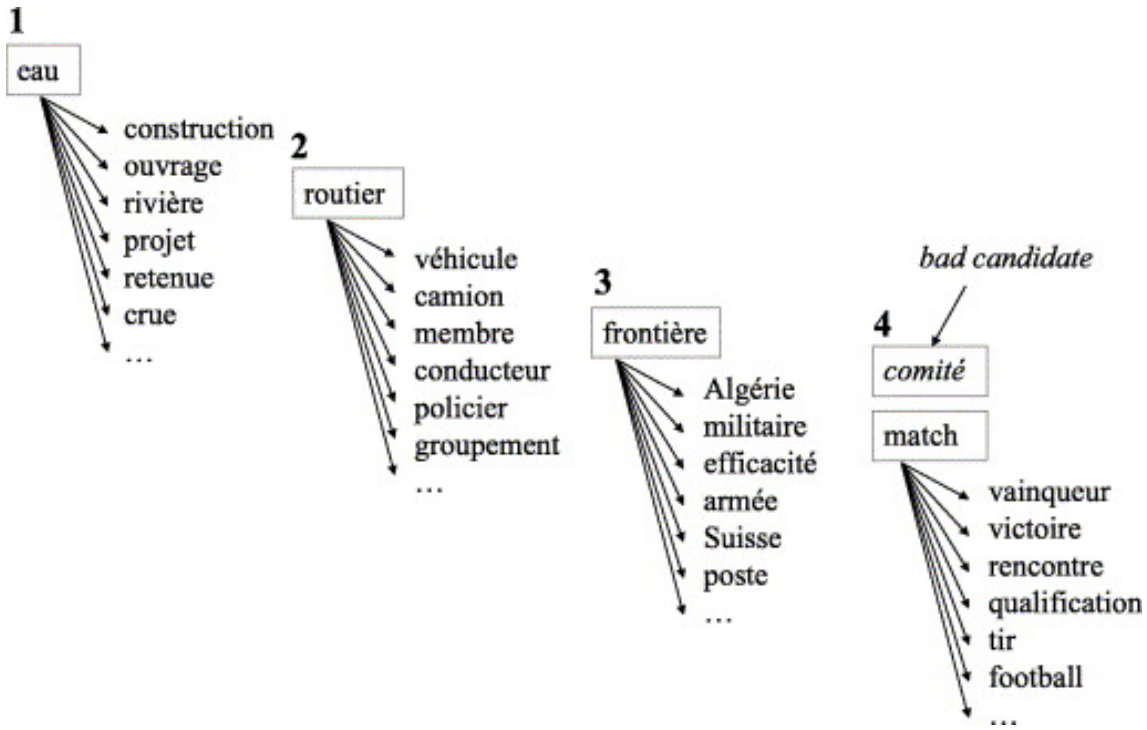

Figure 1: Step-by-step deletion of neighbours

6.2 Detection of root hubs

Input: G: cooccurrence graph;
$Freq$: array of frequencies of nodes in G
$V \leftarrow$ array of nodes in G sorted in decreasing order of frequency;
$H \leftarrow \emptyset$;
while $V \neq \emptyset$ ***et*** $Freq(V[0]) < threshold$ **do**
  $v \leftarrow V[0]$;
  if $GoodCandidate(v)$ **then**
    $H \leftarrow H \cup v$;
    $V \leftarrow V - (v \cup \Gamma(v))$;
  end
end
return H

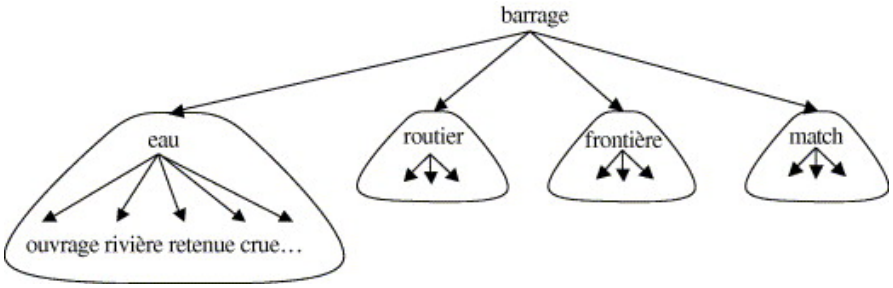

Figure 2: Minimum spanning tree and high-density components.

6.3 Delineating components

Input: G: cooccurrence graph;
H: set of root hubs
t: target word
$G' \leftarrow G \cup t$;
foreach $h \in H$ **do**
| add edge $< t, h >$ with weight 0 to G'
end
$T \leftarrow \mathrm{MST}(G', t)$
return T

6.4 Disambiguation

$$\mathbf{s_i} = \frac{1}{1 + d(h_{i,v})} \text{ if } v \text{ belongs to component } i,$$

$$\mathbf{s_i} = 0 \text{ otherwise.}$$

7 Implementation details

The implementation of the application is broken down into 5 phases 1. File Processing 2. First level (hashing) 3. Second Level 4. Graph processing 5. Disambiguation

7.1 Input data and conll files

The application is run on a cluster with varying number of cores and data to get the expermental results for strong scaling weak scaling and we assume that the application is scalable upto terabytes of input data. The input data for the application are conll format[link to conll paper] files which is shown in the figure ... Each sentence is assigned an id. The words in each sentence are annotated with their lemma or base word and their parts of speech. As mentioned in the hyperlex paper only the adjectives and nouns are considered for disambiguation and all the words which are occuring less than 10 times in the whole data are removed and the ones which are co-occuring less than 5 times are also not considered.

7.2 File Processing

MPI IO provides simultaneous parallel access to a file by a group of processes. The basic MPI::File::seek and MPI::File::read function work much like their unix counterparts. They are independent operations that use local file pointers to determine where in the file to access data for each process. While reading the files the processes a chunk of data from the file with broken sentences or incomplete sentences on either side of the data. we fixed this by sending the broken sentences to the corresponding processes. Once the processes have all complete sentences they proceed for processing the data, in which they extract the words along with their parts of speech, their frequency and their co-occuring words.

7.3 Compression and Decompression

In order to reduce memory latency, the files are read only once and stored in compressed format in the memory. So that they can be retrieved faster and worked upon. The files are compressed by upto 2

7.4 First level

After the initial processing of the data that is read from the files, it so happens that the same words are present in multiple process with partial attribute data like frequency and the co-occuring words. To continue to further processing each word along with all its attribute data must be present in a single process. Initially we broadcasted the words present in each process to every other process and in case the other processes has the same words they sent back the words attribute data and deleted that data in their memory. And this was done linearly from process with rank 0 to n-1. But this resulted in an uneven distribution of data where the first few processes contained large amounts of data and it declined exponentially towards the last process. We bettered this approach by doing in a round robin fashion instead of doing it linearly and by sending only a pre-determined amount of words at a time. But this approach was much inefficient as it increased the amount of communication involved between the processes and thereby increasing the time taken to process.

Keeping in mind, the amount of communication involved between processes and an even distribution of data, we came up with a hash distribution technique where we assign a word to a process depending on their hash value. We calculate the hash of the word string and divides it with MPI communicator size 'n-1' and the remainder is the processes rank which is responsible for this word. Each process can individually calculate the hash for the words they have and send them to the process which is responsible for that word, resulting in a push mechanism where the communication is reduced by half. Fig. ... shows the distribution of the number of words among various processes and it shows that it is almost an even distribution of data.

Figure **??** shows two pictures side by side with one caption.

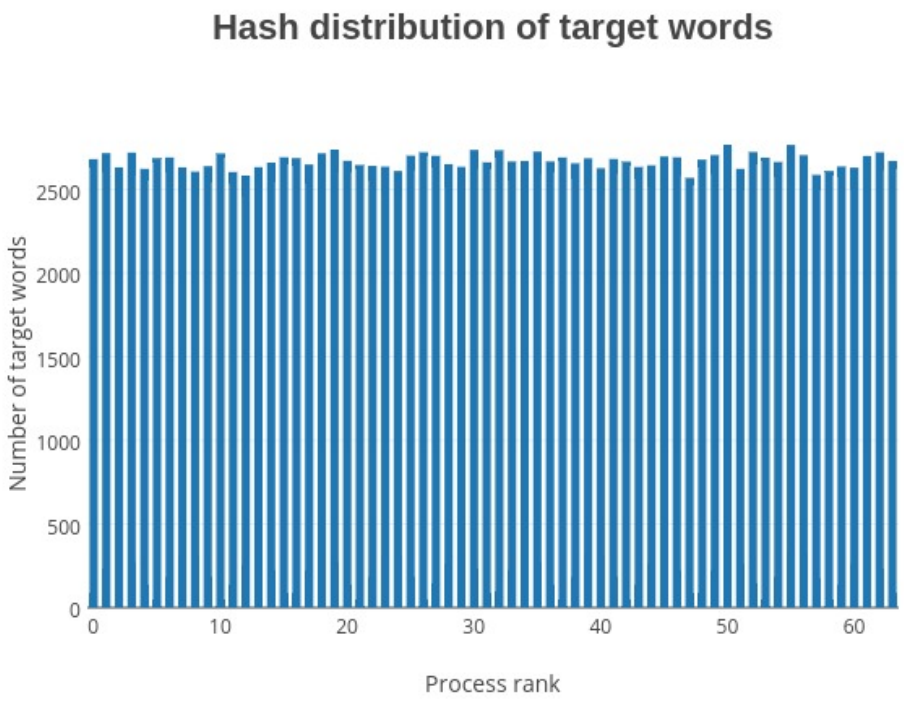

Figure 3: Hash Distribution

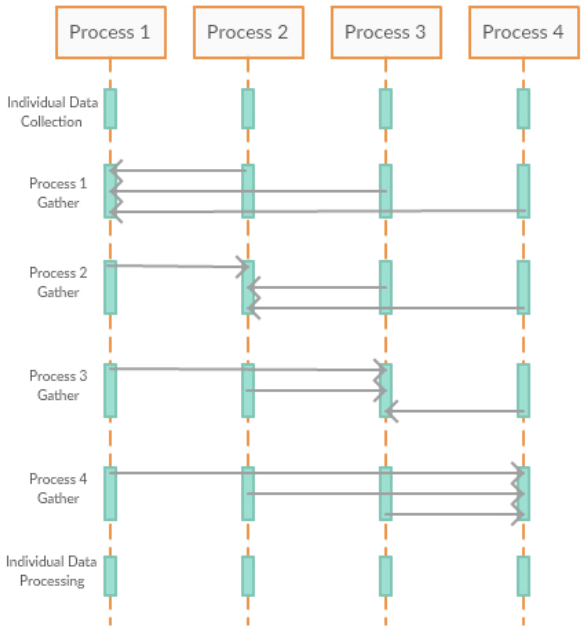

Figure 4: First Level Sequence Diagram

7.5 Second level

To form a graph for a target word, we need all the co-occuring words(first level words) and also the edges between the co-occuring words. There is an edge between two co-occuring words only if the two words are co-occuring themselves. So we need to get the co-occuring words(second level words) of the co-occuring words of the target word to be able to construct a graph.

7.6 Graph Creation and WSI

Once we have the data regarding the first level words along with all the edges between these words, we construct the graph and the roothubs are extracted from the graph. The roothubs are the words which are more frequently co-occuring with the target word. These root hubs form the word senses of the target word. Once we extract the root hubs, we perform an MST on the graph to delineate the words(in other words to align each word to its corresponding roothub), and the distance from itself to its corresponding roothub becomes its score which is later used in disambiguation phase.

7.7 Disambiguation

The

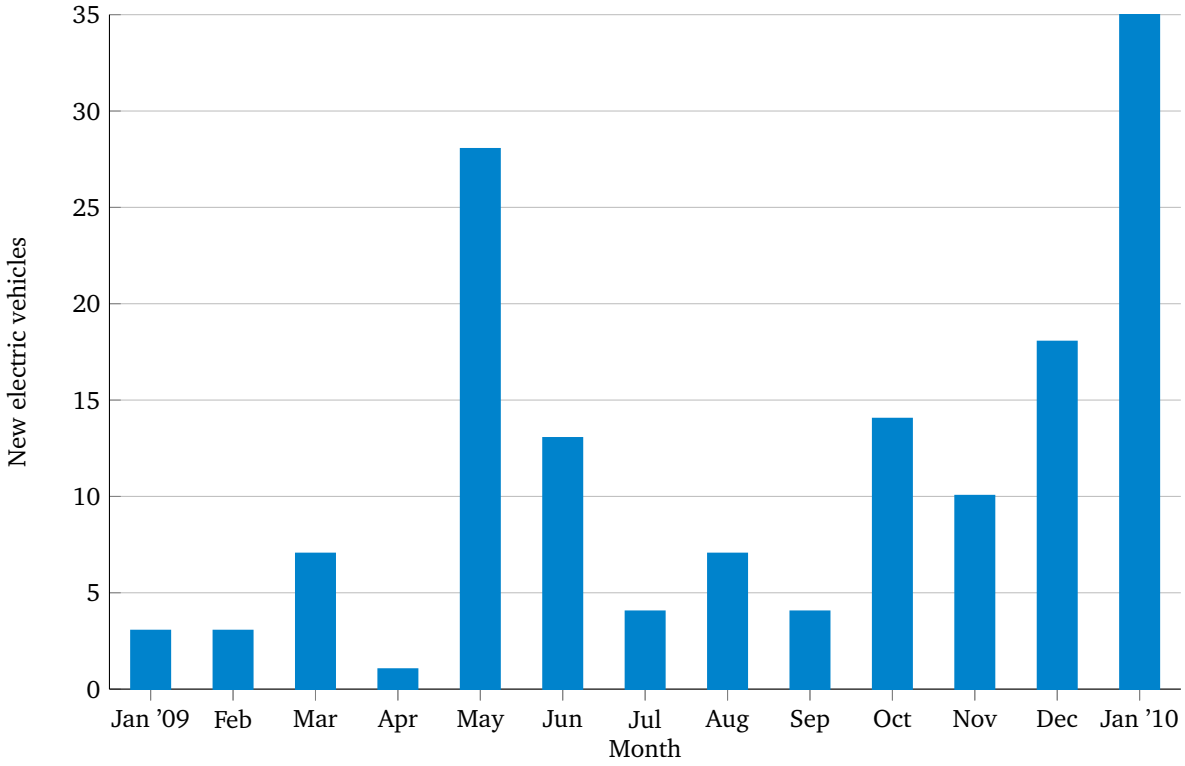

Figure 5: New electric vehicles between Januar 2010 and Januar 2011 [?]