Noname manuscript No.
(will be inserted by the editor)

Tovel: Distributed Graph Clustering for Word Sense Disambiguation

Alessio Guerrieri, Fatemeh Rahimian, Sarunas Girdzijauskas ,Alberto Montresor

the date of receipt and acceptance should be inserted later

Abstract Word sense disambiguation is a fundamental problem in natural language processing (NLP). In this problem, a large corpus of documents contains mentions to well-known (non-ambiguous) words, together with mentions to ambiguous ones. The goal is to compute a clustering of the corpus, such that documents that refer to the same meaning appear in the same cluster; subsequentially, each cluster is assigned to a different semantic meaning. In this paper, we propose a mechanism for word sense disambiguation based on distributed graph clustering that is incremental in nature and can scale to big data. A novel, heuristic vertex-centric algorithm based on the metaphor of the water cycle is used to cluster the graph. Our approach is evaluated on real datasets in both centralized and decentralized environments.

1 Introduction

Language has been ambiguous from its inception. Not only words have several meanings, but even given names (assigned to individuals, companies, or even cities) can be ambiguous. While Ulysses was able to use this feature to his advantage, ambiguity has created more harm than good. This problem has important consequences for web intelligence companies that want to extract public opinions and reaction to news and products from massive data sets acquired by mining social web. Are users complaining about Apple's new phone or about apples that they ate for lunch?

Companies have similar issues when they need to reconciliate their own data with user-input data or different data sources. Understanding the correct meaning for ambiguous words can help in cleaning their datasets, in correcting typos and small mistakes and assigning them to the right category.

This process has become even more cumbersome with the continuous growth in the amount of available data, particularly user-generated content, that makes it extremely important to have automated systems that can both scale to huge sizes and cope with continuous streams of data.

Address(es) of author(s) should be given

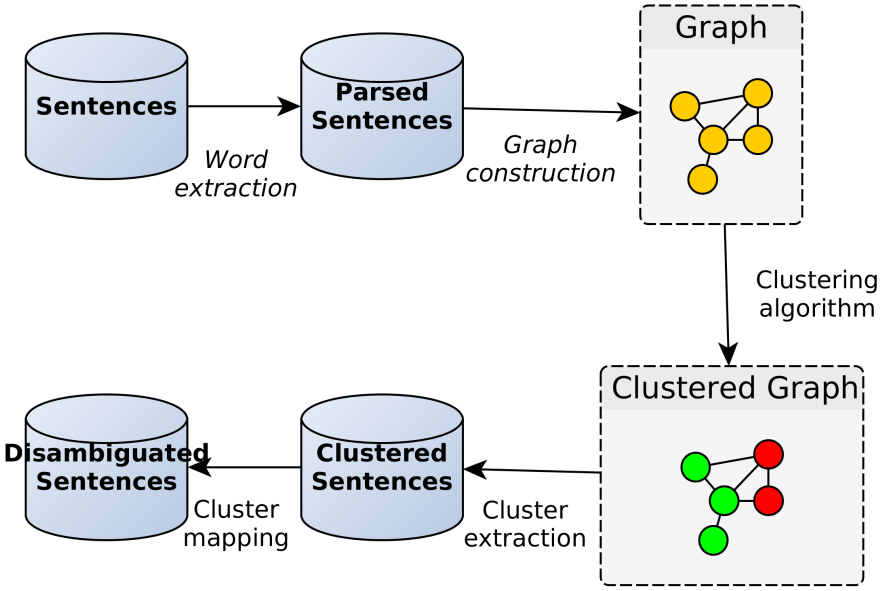


Fig. 1: The pipeline of our approach: from a set of sentences we extract a few words, we built a graph, we cluster it and eventually we map the obtained clusters of sentences to the target meanings.

Many different techniques have been studied in the state of the art [18]. We propose here the following novel approach: (*i*) We take as input a set of documents to be disambiguated, where the important *words* (nouns, verbs, adjectives) have been identified; one target word is *ambiguous* and needs to be disambiguated (Section 2). (*ii*) We build a co-occurrence graph, where words are nodes and two nodes share an edge if they occur in the same document (Section 3). (*iii*) Using our novel distributed graph clustering algorithm, TOVEL, we cluster together the documents that refer to the same ambiguous word (Section 4). (*iv*) All these steps are incrementally executed on incoming data by continuously adding nodes and edges on the ever-evolving graph without having to restart TOVEL from scratch(Section 6).

As shown in Section 7, TOVEL obtains very good results in terms of precision and recall, and outperforms existing approaches in terms of F1-score. Since in TOVEL all nodes act independently, our approach can be scaled to huge quantities of data by implementing it in one of the many distributed large scale graph processing frameworks, such as Giraph, GraphX or Graphlab. We thus demonstrate the scalability of TOVEL using GraphX in Section 7.

2 Problem statement

The problem of finding the correct meaning of a word can be defined in different ways, according to the specific requirements of the problem. The most common defi-

nition for *word sense disambiguation* asks for identification of the correct meaning of a word from a given set of possible meaning. In this chapter we solve another, related problem: *word sense induction*, in which words usages must be grouped according to their shared meaning. Our approach can be extended to word sense disambiguation by using a semi-supervised approach where a few data points are already disambiguated.

We can define the *word sense disambiguation* more formally: given a single *ambiguous word* W, a collection of possible *senses* or *meanings* m_1, m_2, ..., m_M associated with W and a set of *documents* $D = \{d_1, d_2, \ldots, d_N\}$ containing mentions to W, this task asks to understand which of the documents in D refer to the different sense of W. The number of documents N is usually larger than the number of senses M. Our approach is based on the following sub-problems:

- *Clustering*: compute a clustering of D such that documents that refer to the same meaning appear in the same cluster.
- *Disambiguation*: assign each document in D to one of the meaning of W, according to clustering obtained in the previous step.

If the algorithm stops after the Clustering phase, the problem solved is *word sense induction*, but the addition of the mapping step solves the *word sense disambiguation*. Table 1 contains an illustrative toy instance of our problem. The word "apple" can refer to at least two different meaning: the fruit apple and the company. To solve the clustering subproblem would mean to recognize that documents 1 and 4 refer to one meaning, while 2 and 3 refer to a different one. To complete the disambiguation subproblem we also need to map each of the document to a specific meaning (Apple Inc. for documents 2 and 3, apple the fruit for documents 1 and 4). Note that other possible meanings of Apple (such as the Beatles' multimedia corporation) are irrelevant to this instance of the problem, since there are no documents that refer to it.

3 Graph construction

The first step of our approach is the construction of the word graph from the input documents. The graph construction follows the same principles of [26] and is illustrated in Algorithm 1. For each document d, we add a distinct *ambiguous node* representing the ambiguous mention of the target word W, identified by the document

#	Sentence
1	Many doctors would recommend eating **apples** for breakfast
2	Technology like **apple**'s watches could help doctors monitor their patients
3	**Apple** produces many distinct technologies, from smartphones to watches
4	Before eating an **apple**, I always check which is its variety.

Table 1: Simple example: 4 documents, *Apple* is the target ambiguous word with two different senses (the fruit and Apple Inc.)

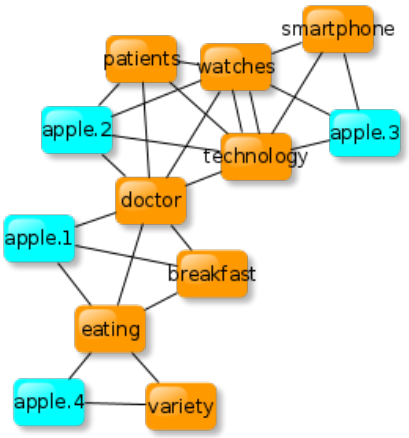


Fig. 2: Sample graph created from Table 1. Blue nodes are ambiguous.

id; we create a new node for each context word w extracted from d, unless it already exists in the graph. The nodes representing the words contained in the document (either created or found through getNode()) are added to the set S; then, undirected edges are created between all pair of distinct nodes contained in S, with the effect of creating a clique among them.

For the sake of simplifying the notation and save space, in this paper G is a multigraph where parallel edges between pair of nodes correspond to stronger links between them. In the real implementation, edges are weighted and the interactions among nodes connected by edges are proportional to such weights.

The strategy for extracting context words from the document depends on the specific application area. In our current approach on text documents, we just select every noun and adjective from surrounding sentences, converting everything to lowercase. Note that this step is needed only when we need to disambiguate plain text. In some

Algorithm 1: Graph construction

```
G = new GRAPH()
foreach DOCUMENT d do
    NODE ambNode = new NODE(d.id)
    G.addNode(ambNode)
    SET S = {ambNode}
    foreach WORD w ∈ d do
        if not G.contains(w) then
            G.addNode(new NODE(w))
        S = S ∪ G.getNode(w)
    foreach u, v ∈ S, u ≠ v do
        G.addEdge(u, v)
        G.addEdge(v, u)
```

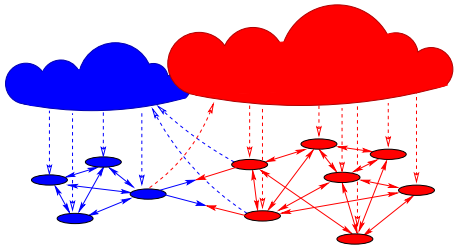


Fig. 3: Illustration of the water cycle in TOVEL

applications, such as the Spotify scenario presented in Section 7, the documents are already provided as a bag of context words.

In Table 1, all context words are underlined, while mentions of our target word are in bold. Figure 2 shows the graph constructed from this example. Aside from the blue nodes, one for each document, all other nodes are shared across the documents. Note that context words "watches" and "technology" co-occur in two different documents, thus increasing the strength of their connection. In a real implementation, we would have created an edge of weight 2 to connect them.

4 Tovel: a Distributed Graph Clustering Algorithm

Our novel clustering algorithm, TOVEL, is inspired by the cycle of water. Water of different colors is generated at initialization and competes for control of the graph through a cycle of diffusion, evaporation and rain. Some colors will disappear from the graph, while others will survive by following the shapes of the underlying clusters. The resulting clusters represent the different meanings of the ambiguous word.

Using the input graph constructed as in Section 3, we create one color for each node representing the ambiguous word. This means that initially, the number of colors will be equal to the number of documents. This number is appropriate because we are not expecting more meanings than documents.

Note that, since the algorithm uses information about ambiguous nodes only at initialization, TOVEL could also be used on a general graph by starting with a reasonable number of colors from random starting position.

The algorithm can be summarized as following: during each iteration, each node diffuses its water to its neighbors. Each node will then decide independently its dominant color according to information in the neighborhood. All water of non-dominant colors is evaporated and sent to the appropriate cloud, one for each color in the graph. Each cloud will then try to send back POUR of its water to all nodes with its dominant color, if enough water is present in it. The remaining water will be kept in the clouds for the following iterations. Once the algorithm has converged to a solution, all nodes with the same dominant color will form a cluster.

Algorithm 2: Initialization phase

```
{ Executed by node u ∈ V }
if isAmbiguous() then
    u.color = u.id
    u.water = 1.0
else
    u.color = Nil
    u.water = 0.0
H = new MAP()
```

4.1 Data structures

Every node u contains a variable $u.color$ that represents the *dominant* color of that node and a variable $u.water$ containing the amount of water of color $u.color$ contained in u. Furthermore, a map H will be used to collect colors diffused from other nodes.

Apart from the set of nodes, we create a set of *clouds* $C_1, C_2, \ldots, C_n$, one for each color and thus one for each document. Each cloud C_i contains an amount of water $C_i.water$ of color i. As shown in Algorithm 2, each ambiguous node starts with a fixed amount of an unique color, identified by the unique id of the node. The non-ambiguous nodes and the clouds start empty. This is the only point in the algorithm where water is created.

4.2 Main cycle

TOVEL is organized in consecutive rounds, each of them subdivided in three independent phases. In the first phase (*diffusion*), each node diffuses the water of its dominant color by sending it through each of its edges, divided equally among all of them. In the second phase (*evaporation*), each node computes a new dominant color and evaporates all the water of non-dominant colors by sending it to the clouds. In the third phase (*raining*), all cloud sends their water back to the nodes with their dominant color. An upper bound POUR is used to limit the rate of rain in each round.

Algorithm 3: Diffusion phase

```
{ Executed by node u ∈ V }
foreach NODE v ∈ u.neighbors do
    real amount = u.water/u.degree
    send ⟨u.color, amount⟩ to v

H.clean()
foreach NODE v ∈ u.neighbors do
    receive ⟨v.color, amount⟩ from v
    H[v.color] = H[v.color] + amount
```

Algorithm 4: Evaporation phase

```
{ Executed by node u ∈ V }
foreach NODE v ∈ u.neighbors do
    send⟨H⟩ to v

MAP H' = H.copy()
foreach NODE v ∈ u.neighbors do
    receive ⟨H_v⟩ from v
    foreach COLOR c ∈ H_v do
        H'[c] = H'[c] + H_v[c]
u.color = argmax(H'(c))
u.water = H[color]
H[u.color] = 0
foreach COLOR c do
    send ⟨H[c]⟩ to C_c
```

Diffuse Each node in the graph sends all its water to its neighbors, divided equally among the (multi)edges connecting them. For example, if a node has 0.8 of red water and 4 outgoing edges, it will send 0.2 of red water along each of them. If two edges among those are pointing to the same node, that node will receive 0.4 of red water. The strengths of connections will influence the behavior of the diffusion process. Nodes will then wait until they receive messages from each of their neighbors, and aggregate the amount of water received in a fresh map indexed by colors.

Evaporation Each node recomputes its dominant color by summing all the maps of its neighbors and choosing the color with the highest amount. In case of ties, nodes will choose the color with the lowest id. We chose this simple and deterministic heuristic since other approaches, such as breaking ties randomly, did not improve the quality of the clustering in our experiments. Computing the dominant color is the most computationally expensive step, but it allows us to better understand the shape of the cluster by looking at our neighbor's neighbors [15]. By collecting the colors of our neighbor we can glimpse at what we will receive in the following iteration and choose how to interact with the clouds accordingly. Each node then sends all water of non-dominant color to the appropriate clouds. If a node has chosen red as its dominant color, it will send all of its blue color to the blue cloud.

Rain Each cloud receives water sent by the nodes in the graph and sums it with all water it kept since the previous iteration. It will then send some water back to the nodes following this procedure: the cloud of color c will compute the set of nodes that have c as dominant color; it will then try to send at most POUR amount of water to them. If the cloud does not contain the necessary amount of water, it will just divide it equally between the nodes with that color. If there are no nodes of color c in the graph, the corresponding cloud will not be able to send water to any node and the color will thus disappear from the graph.

Algorithm 5: Rain phase

```
{Executed by cloud C_c}
foreach NODE u ∈ V do
    receive ⟨amount⟩ from u
    C_c.water = C_c.water + amount
SET S = {u : u ∈ V ∧ u.color = c}
if |S| > 0 then
    rain = min(POUR, C_c.water/|S|)
    foreach NODE u ∈ S do
        send ⟨rain⟩ to u
    C_c.water = C_c.water − rain ∗ |S|
{Executed by node u}
receive ⟨amount⟩ from C_{u.color}
u.water = u.water + amount
```

4.3 Convergence criterion

Each node will vote to halt the algorithm if its dominant color has not changed for a sufficient number of iterations. The algorithm is stopped when all nodes vote to terminate. In our experiments, we call this parameter IDLE and set its value to 5. A larger value does not increase significantly the quality of our clustering and come at the cost of a much larger iteration count.

While this simple approach has been sufficient to converge with real-world graphs, there are corner cases in which convergence is never reached because of a flickering effect in which a few nodes continuously switch between colors.

4.4 Rationale

TOVEL is an heuristic approach to an NP-complete problem. In this section we provide an overview of the rationale behind the different phases for our approach, while Section 5 presents a more analytical study of the quality of the algorithm and its relation to the conductance of the graph.

Figure 3 illustrates the cycle of red water in the algorithm. The red cluster will diffuse the red water along both red and blue edges. All water that is sent towards blue nodes will go out of the red cluster and, if it does not convince those blue nodes to change their color, will be evaporated and sent back to the red cloud. The cloud will then rain some of the red water back to the cluster.

We call the *leakage* of a cluster the amount of water lost by that cluster via the process of diffusion and evaporation, and the *precipitation* of that cluster the amount of water that it has received from the cloud.

It is easy to see that if the leakage is larger than the precipitation, the total amount of water in the cluster will decrease. This process will also decrease the average amount of water in the cluster, thus decreasing the leakage in the following iterations. If we assume that there is an infinite amount of water in the cloud, the precipitation

V_c	set of nodes with dominant color c
deg_n	degree of node n
cut_n	edges between node n and nodes of a different color
W_c	total amount of water c in the graph
$\overline{W_c} = W_c/\|V_c\|$	average amount of water c in the nodes of its cluster

Table 2: Notations used in the analysis of TOVEL

will stay constant if the cluster does not change and the leakage will eventually be equal to the precipitation. If the leakage is smaller than the precipitation, the inverse process will appear and the leakage will grow until it reaches the precipitation.

The core property of the algorithm is the following: the smaller is the fraction of outgoing edges of a cluster (close to its conductance), more water will each node of the cluster have at the converged state. This property is crucial in making sure that well-connected clusters survive the competition, while badly connected clusters will be weaker, get invaded more easily and eventually disappear.

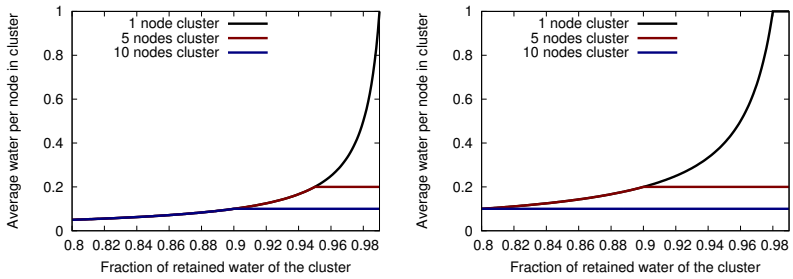
The total quantity of water in TOVEL is fixed, thus it is possible that a cloud will not be able to send the full POUR to the nodes of the clusters. This event becomes more likely once a cluster gets bigger, since it will spread its fixed amount of water on a larger set of nodes. By choosing the correct POUR, we can thus control the desired sizes of the clusters. In our experiments, we used an heuristic based on the ratio of distinct context words over total context words, as illustrated in Section 5

5 Analysis

In this section we analytically study the behavior of TOVEL and show that it will tend to favor well-structured subgraphs of the desired size. The notation used is defined in Table 2.

5.1 Quality at convergence

To help our analysis, we will analyze the behavior of TOVEL once it has reached the steady state and the clusters are fixed. Each cluster will diffuse some of its water to neighbors of a different color, who will then send it back to the cloud. The total amount of water c that evaporate to the cloud in each round can be computed as in Equation 1. Each node of the cluster has a certain amount of water of that color and a fraction of it will be sent to neighbors of a different color. Assuming that the distribution of water inside each cluster is uniform we can continue the analysis and obtain Equation 2. During each iteration the cluster also receives some water from the clouds. If there is sufficient water there, each node will receive exactly POUR of water of its dominant color. We reach the steady state when the precipitation and the leakage are the same. How much water will there be in each node of the cluster at that point? By solving Equation 4 we obtain a value for the average amount of water as in Equation 5



(a) POUR = 0.01 (b) POUR = 0.02

Fig. 4: Average color per node in a cluster at steady state, for different quality and values of POUR, in a graph with 1000 nodes

$$Leakage_c = \sum_{n \in V_c} W_n(c) \frac{cut_n}{deg_n} \tag{1}$$

$$Leakage_c = \sum_{n \in V_c} \overline{COL_c} \frac{cut_n}{deg_n} = \overline{COL_c} \sum_{n \in V_c} \frac{cut_n}{deg_n} \tag{2}$$

$$Precipitation_c = |V_c| * \text{POUR} \tag{3}$$

$$|V_c| * \text{POUR} = \overline{COL_c} \sum_{n \in V_c} \frac{cut_n}{deg_n} \tag{4}$$

$$\overline{COL_c} = \frac{\text{POUR} \cdot |V_c|}{\sum_{n \in V_c} \frac{cut_n}{deg_n}} \tag{5}$$

This formula shows that the average amount of water contained in a node in the steady state depends on the average fraction of water that evaporates during each iteration. This measure is very close to the conductance of the cluster, since the fewer cut edges there are, the bigger will be the average amount of color in the nodes of that cluster. This is not only true at the steady state, but also during the execution of the algorithm. Badly formed clusters will see its color evaporate much faster and will be made easier to invade by the other clusters.

5.2 Sizes at convergence

The feature that allows us to control the sizes of the partitions is the fixed amount of water in the system. The clusters are discouraged from becoming too large because the average amount of color cannot be more than $\frac{1}{|V_C|}$.

Figure 4 illustrates the effect of different values of POUR. It shows the average amount of water in each node of the cluster, a measure of the strength of the cluster in our algorithm, against the fraction of water that is kept during each iteration by

the cluster. If a cluster contains 0.8 of water and loses 0.2 because of evaporation, it means that it keeps 0.75 of its water.

Figure 4b shows that clusters of size smaller than 5 and quality higher than 0.9 will keep more color than clusters of size 5 with the same quality. The higher the size, the more of its clusters will be penalized. Choosing a different POUR, as shown in Figure 4a, changes the strength of that effect. Fewer clusters of size 5 will be penalized, but the behavior of clusters of size 10 is now close to the behavior of clusters of size 5 with the previous value of the cap. By choosing and tuning the value of this parameter we can control the desired sizes of the clusters and encourage smaller but less connected clusters or larger and better connected clusters.

5.3 Computing POUR for word sense induction

While in a general graph we do not have much information about the underlying clusters, in the word sense induction scenario we start from a better position. The graph is created as a collection of cliques, one for each document, and is thus possible to estimate how much dense is the graph and thus how conservative the algorithm should be in creating clusters. If the graph is not dense, then POUR can be set to a lower value, while if the graph is very dense, an higher POUR might allow us to find clusters with lower quality, but still meaningful, thus avoiding finding only one huge cluster.

In this subsection we present an heuristic to set POUR in a meaningful way, while still allowing users the freedom to control the degree of resolution of the cluster. In Section 7 we show that this approach allow us to obtain very good results with graphs of wildly different characteristics.

Be A the set of ambiguous nodes in the graph and $\overline{A} = N \setminus A$ the set of non-ambiguous nodes in the graph. The following measure is an indication of the density of the graph:

$$d = \frac{\text{total context words extracted from documents}}{\text{distinct context words extracted from documents}} = \frac{\sum_{n \in A} deg_n}{|\overline{A}|}$$

Note that each ambiguous node represents a document, and its degree is equal to the number of context words associated to that document. The numerator of the fraction is thus equal to the total number of context words in the dataset. The denominator is instead equal to the number of distinct context words, since for each of them we create only one word (see Section 3).

$$\text{POUR} = \frac{d}{|N|}$$

This metric is strongly related to the eventual size of the clusters that will be found by our approach. If the graph is very sparse, then POUR will be equal to $\frac{1}{|N|}$, thus indicating that a full cloud is able to serve clusters size at most $|N|$, the largest cluster possible. If, instead, the graph is very dense, each ambiguous node will be

Algorithm 6: Classifier

```
Input  : Sentence S
Input  : Colored graph G
M = new MAP();
foreach WORD w ∈ S do
    if w ∈ G then
        color = G.getNode(w).color;
        amount = G.getNode(w).water;
        M(col) = M(col) + amount;
return arg max_c(M(c));
```

connected to the same context words and POUR will be equal to $\frac{|A|}{|N|}$. A full cloud is thus able to serve a clusters of size $\frac{|N|}{|A|}$, which is the average size of a cluster if we create one different cluster for each ambiguous node. Since we do not care about clusters that do not contain ambiguous nodes for this specific application, this is the smallest cluster size we are interested in.

6 Extensions for word sense induction and disambiguation

Incremental addition of batches of documents Since our approach should be run on huge quantities of documents, it is infeasible to run it from scratch every time there is an update in the dataset. For this reason, both the graph construction and the graph clustering algorithms can be adapted to work in an incremental scenario.

If we assume that the new batch of sentences to be added does not introduce new meanings (does not change the number of clusters in the ground truth), then it is possible to extract the context words from the new sentences according to Section 3 and add any newly created node to the graph without any water, while keeping the old distribution of colors in the rest of the graph. TOVEL will converge extremely fast since it will start from a state already close to the desired result.

Colored graph as a classifier The incremental algorithm can be used when we need to continuously update our inner model, but in some cases we might want to run our approach only once on a large dataset and then use that model to answer queries on single input sentences independently. By following this approach, we get huge gains in efficiency and scalability, since the colored graph can be accessed independently for each query in "read-only mode", but we lose the capability of use the input sentences as part of our dataset.

Given the colored graph, we assign each color to a meaning of the target word by manually disambiguating a few sample sentences in the dataset. Once we have a mapping between the colors and the meanings, we store for each non-ambiguous word both its color and the amount of water of that color. For each input sentence we extract its context words and, if they exists in our colored graph, collect all water that they contain. The input sentence will be classified according to the most popular color, computed following Algorithm 6.

Dataset statistics						TOVEL results				
Name	Source	Docs	Nodes	Edges	Senses	Prec	Rec	F1	F05	Rounds
Apple	Wikipedia	369	5046	258798	2	91.61	85.52	88.46	90.33	42
Mercury	Wikipedia	2921	15111	816744	4	86.52	80.40	83.35	85.22	40
Orange	Wikipedia	1447	9546	489736	5	74.23	61.21	67.09	71.20	46
CA	Recorded Future	1182	2045	16700	8	97.92	68.83	80.84	90.29	43
Kent	Spotify	124	160	1654	6	95,76	97,64	96,69	96,13	10

Table 3: Datasets used in evaluation

This approach could also be applied on a training set of sentences of which the correct meaning is already known. The resulting colored graph could then be used to disambiguate all other sentences.

7 Experimental results

In this study we used datasets collected from different sources. *Apple*, *Mercury* and *Orange* are taken from Wikipedia. For each meaning of that word, we extracted sentences with outlinks toward that page. The context words are automatically extracted by selecting all adjectives and nouns using the Stanford NLP parser tool. *CA* contains a set of documents pertaining different "Chris Andersen" as collected by Recorded Future, a web intelligence company. The words are extracted from each document using their proprietary techniques. *Kent* was constructed by Spotify from a collection of albums with the field "Artist Name" equal to "Kent". The context words extracted from the albums are other fields such as the recording company, the country and the language. This graph is much smaller but shows the feasibility of our approach in scenarios different from text disambiguation, such as artist disambiguation. Both *CA* and *Kent* have been confidentially given to us by the respective companies. In the case of *Kent*, our algorithm is already used as a pre-processing phase in Spotify's system.

To evaluate the quality of our clustering we use the B-cubed approach to compute the precision, recall and F1-score of the ambiguous nodes, as presented in [3]. High precision means clusters that are clean and contain nodes that have the same meaning, while high recall means having clusters that contain most nodes of that meaning. The F05-score gives twice the importance to precision than recall. In Table 3 we show the performance of our approach.

Evaluation on benchmarks We executed our approach on the Semeval-2010 Word Sense Induction Task and compared our clusters with the published results from the competition in Table 5

Clustering comparison with disambiguation services To give a comparison of the quality of the clustering of our approach, we used the NERD API to run different disambiguation services on our own datasets. Table 4 shows that our approach reaches results comparable with the leading disambiguation services, without using any external source.

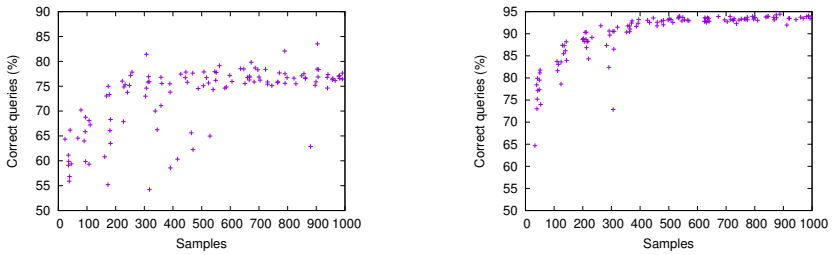
Disambiguation with colored graph To simulate our approach in a realistic scenario, we follow the model from Section 6. From a random subset of our dataset we build the graph, run the clustering algorithm and assign each cluster to a meaning. We

Service	Prec	Rec	F1	F05	
textrazor	100.00	79.17	88.38	95.00	Apple
dbpedia	98.94	76.07	86.01	93.33	
wikimeta	96.29	69.41	80.67	89.37	
combined	98.45	54.15	69.87	84.61	
TOVEL	91.61	85.52	88.46	90.33	
textrazor	74.23	27.10	39.71	55.08	Mercury
dbpedia	75.26	28.35	41.19	56.55	
wikimeta	73.56	27.23	39.74	54.88	
combined	97.29	53.51	69.04	83.61	
TOVEL	86.52	80.40	83.35	85.22	

Table 4: Comparison of our approach against online disambiguation services

System	F-Score			Clusters Number
	All	Verbs	Nouns	
MFS	63.4	72.7	57.0	1
Duluth-WSI-SVD-Gap	63.3	72.4	57.0	1.02
Tovel	***63.0***	***72.0***	***56.9***	***1.81***
KCDC-PT	61.8	69.7	56.4	1.5
KCDC-GD	59.2	70.0	51.6	2.78
Duluth-Mix-Gap	59.1	65.8	54.5	1.61

Table 5: Comparison of our approach against top competitors in Semeval-2010



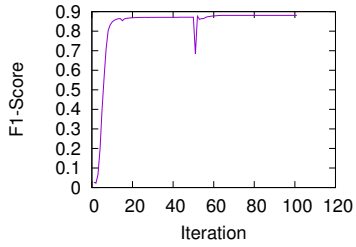
(a) Mercury dataset

(b) Apple dataset

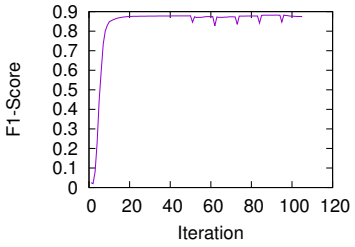
Fig. 5: Percentage of queried sentence correctly classified, against the number of sampled sentences used in the model construction.

then use this classifier to process the remaining sentences. As Figure 5 illustrates, our approach can disambiguate with high precision once the size of the learning dataset is large enough.

Incremental results Figure 6 shows the behavior of our algorithm in presence of incremental updates in the graph. We test two different scenarios: in the first scenario our approach is run to convergence on 80% of the document and the remaining 20% are added in a single batch after 50 iterations. In the second scenario our approach is run on 50% of the graph and 5 batches of 10% each are added at specific interval. In both cases the algorithms takes only a few iterations to recover and reach convergence.



(a) Starts with 80% of documents, adds 20% after 50 iterations



(b) Starts with 50% of documents, after 50 iterations adds 10% each, 5 times

Fig. 6: F-score against iteration

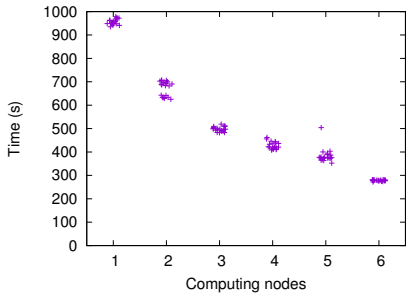


Fig. 7: Running time against number of machines of spark implementation

Scalability TOVEL is extremely scalable, since each vertex and each cloud can work in parallel in each of the different stages of Tovel. Figure 7 shows the running time of our prototype implementation in Spark on the EC2 cloud, using different number of nodes.

8 Related work

Our approach is based on using graph clustering algorithms for word sense induction and disambiguation. Therefore, the related work introduces both traditional word sense disambiguation algorithms and an overview of graph clustering algorithms.

8.1 Word sense induction and disambiguation

There is a large body of work on word sense disambiguation in the NLP community [19]. The problem is typically seen as a classification task, where different senses of a word are classified into different generic classes. One of the most well-known approach is by Lesk et al. [14], which computed the size of overlap between the glosses of the target and the context words, as an indication for classification. Since

then various efforts has been made to extend the original Lesk algorithm [13,4,5]. An inherent limitation in the Lesk algorithm, however, is that the results are highly sensitive to the exact wordings of the definition of senses [19]. To overcome this problem, some solutions computed the overlap differently. For example, Chen et.al [9] used tree-matching, and recently various solutions used vector space models [23,1,27]. They, however, struggle with the problem at large scale. TOVEL computes the overlap between various context words using a graph model, without having to concern about grammatical or syntactical properties, while it addresses the scalability problem with a highly parallel algorithm.

After deciding what information to use, the main task of classification starts. The solutions are either supervised [31,24,28], unsupervised [2,8,29], or semi-supervised. Supervised methods generally produce reasonably accurate results, but the training data is usually not available in the real-world applications. Therefore, semi-supervised and unsupervised methods have gained lots of attention. While unsupervised solutions exploit the dictionary entries or taxonomical hierarchies like WordNet [17], the semi-supervised solutions start with some labeled data as seeds [22].

8.2 Graph clustering

The research in graph community detection itself has produced numerous works [11]. The problem, which is known to be NP-Hard, has been addressed through various heuristic solutions. A few approaches use the spectral properties of the graph for clustering [21], by transforming the initial set of nodes/edges into a set of points in the space, whose coordinates are the element of the eigenvectors of the graph adjacency matrix. The intuition is that nodes that belong to the same community structure have similar components in their eigenvectors. The problem with spectral clustering is that computing eigenvectors for large graphs is non-trivial, thus this solution is not applicable in large scale.

Some other solutions aim for maximizing the modularity metric. This approach, which was first introduced by Girwan and Newman [20], involves iteratively removing links with the highest betweenness centrality, until the maximum modularity is achieved. The problem with this technique is that computing the betweenness centrality is a complex task and requires global knowledge of the graph, thus, it cannot scale in large graphs. Many others extended the modularity optimization idea and made an effort to make it faster and more efficient [10,7]. However, [12] has shown that the modularity has a resolution limit, and therefore, these solutions sometimes show a poor performance, particularly if there is a large graph with lots of small communities.

There are also solutions based on random walks in graphs [25]. The intuition is that random walks are more likely to get trapped in the densely connected regions of a graph, which correspond to the community structures. Other ideas, which are similar to random walk in nature, include the diffusion [16] and label propagation [30] [6] techniques. These solution have shown reasonable performance in general, and for word sense disambiguation in particular [26,22], while having a relatively low com-

plexity. They can be applied to large graphs, thanks to their parallel nature. TOVEL has the closest resemblance to this family of solutions.

9 Conclusions

In this paper, we have proposed a mechanism for word sense disambiguation based on distributed graph clustering that is incremental in nature and can scale to large amount of data. Our approach has shown to be efficient and precise in both centralized and decentralized environments.

As a future work, we intend to study TOVEL in a more general graph clustering scenario. In such a setting we do not have information about ambiguous nodes, therefore we need different approaches to initialize the distribution of water in the graph. Starting with a random sample of nodes might be enough, provided that the number of nodes is reasonably larger than the expected number of communities to be found in the graph.

Our TOVEL implementation of TOVEL is still a proof of concept that needs to be polished to scale efficiently to truly large datasets. Since there are few distributed algorithm to compare with, there is also the option of creating a faster, parallel program to compare its efficiency with other parallel clustering algorithm.

References

1. Khaled Abdalgader and Andrew Skabar. Unsupervised similarity-based word sense disambiguation using context vectors and sentential word importance. *ACM Transactions on Speech and Language Processing (TSLP)*, 9(1):2, 2012.
2. Eneko Agirre, David Martínez, Oier López de Lacalle, and Aitor Soroa. Two graph-based algorithms for state-of-the-art WSD. In *Proc. of the 2006 Conf. on Empirical Methods in Natural Language Processing*, pages 585–593. ACL, 2006.
3. Amit Bagga and Breck Baldwin. Algorithms for scoring coreference chains. In *Proc. of the Linguistic Coreference Workshop at The First International Conference on Language Resources and Evaluation (LREC'98)*, pages 563–566, 1998.
4. Satanjeev Banerjee and Ted Pedersen. An adapted Lesk algorithm for word sense disambiguation using WordNet. In *Computational linguistics and intelligent text processing*, pages 136–145. Springer, 2002.
5. Satanjeev Banerjee and Ted Pedersen. Extended gloss overlaps as a measure of semantic relatedness. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 805–810. Morgan Kaufmann Publishers Inc., 2003.
6. Chris Biemann. Chinese whispers: an efficient graph clustering algorithm and its application to natural language processing problems. In *Proceedings of the 1st Workshop on Graph-based Methods for Natural Language Processing*, pages 73–80, 2006.
7. Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of community hierarchies in large networks. *CoRR*, abs/0803.0476, 2008.
8. Samuel Brody and Mirella Lapata. Bayesian word sense induction. In *Proc. of the 12th Conf. of the European Chapter of the ACLs*, pages 103–111. ACL, 2009.
9. Ping Chen, Wei Ding, Chris Bowes, and David Brown. A fully unsupervised word sense disambiguation method using dependency knowledge. In *Proc. of the 2009 Annual Conf. of the North American Chapter of the ACLs*, pages 28–36. ACL, 2009.
10. Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.
11. Santo Fortunato. Community detection in graphs. *Phys. Rep.*, 486(3):75–174, 2010.

12. Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proc. of the National Academy of Sciences*, 104(1):36–41, 2007.
13. Adam Kilgarriff and Joseph Rosenzweig. Framework and results for english SENSEVAL. *Computers and the Humanities*, 34(1-2):15–48, 2000.
14. Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proc. of the 5th International conference on Systems documentation*, pages 24–26. ACM, 1986.
15. Gurmeet Singh Manku, Moni Naor, and Udi Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized P2P networks. In *Proc. of the 36th ACM symposium on Theory of computing*, pages 54–63. ACM, 2004.
16. Henning Meyerhenke, Burkhard Monien, and Thomas Sauerwald. A new diffusion-based multilevel algorithm for computing graph partitions. *Journal of Parallel and Distributed Computing*, 69(9):750–761, 2009.
17. George A Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.
18. Roberto Navigli. Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)*, 41(2):10, 2009.
19. Roberto Navigli. A quick tour of word sense disambiguation, induction and related approaches. In *SOFSEM 2012: Theory and practice of computer science*, pages 115–129. Springer, 2012.
20. Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.
21. Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.
22. Zheng-Yu Niu, Dong-Hong Ji, and Chew Lim Tan. Word sense disambiguation using label propagation based semi-supervised learning. In *Proc. of the 43rd Annual Meeting on ACLs*, pages 395–402. ACL, 2005.
23. Siddharth Patwardhan and Ted Pedersen. Using WordNet-based context vectors to estimate the semantic relatedness of concepts. In *Proc. of the EACL 2006 Workshop Making Sense of Sense-Bringing Computational Linguistics and Psycholinguistics Together*, volume 1501, pages 1–8, 2006.
24. Mohammad Taher Pilehvar and Roberto Navigli. A large-scale pseudoword-based evaluation framework for state-of-the-art word sense disambiguation. *Computational Linguistics*, 40(4):837–881, 2014.
25. Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. In *Computer and Information Sciences (ISCIS 2005)*, pages 284–293. Springer, 2005.
26. Fatemeh Rahimian, Sarunas Girdzijauskas, and Seif Haridi. Parallel community detection for cross-document coreference. In *IEEE/WIC/ACM International Joint Conf. on Web Intelligence and Intelligent Agent Technologies*, volume 2, pages 46–53. IEEE, 2014.
27. Ariel Raviv, Shaul Markovitch, and Sotirios-Efstathios Maneas. Concept-based approach to word-sense disambiguation. In *AAAI*, 2012.
28. Hui Shen, Razvan Bunescu, and Rada Mihalcea. Coarse to fine grained sense disambiguation in wikipedia. *Proc. of SEM*, pages 22–31, 2013.
29. Tim Van de Cruys and Marianna Apidianaki. Latent semantic word sense induction and disambiguation. In *Proc. of the 49th Annual Meeting of the ACLs: Human Language Technologies-Volume 1*, pages 1476–1485. ACL, 2011.
30. Fei Wang and Changshui Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):55–67, 2008.
31. Zhi Zhong and Hwee Tou Ng. It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proc. of the ACL 2010 System Demonstrations*, pages 78–83. ACL, 2010.