Building Instance Knowledge Network for Word Sense Disambiguation

Shangfeng Hu, Chengfei Liu

Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn 3122, Victoria, Australia
{shu,cliu}@groupwise.swin.edu.au

Xiaohui Zhao

Information Systems Group
Department of Industrial Engineering & Innovation Sciences
Eindhoven University of Technology
Eindhoven, The Netherlands
x.zhao@tue.nl

Marek Kowalkiewicz

SAP Research
Brisbane, Australia
marek.kowalkiewicz@sap.com

Abstract

In this paper, a new high precision focused word sense disambiguation (WSD) approach is proposed, which not only attempts to identify the proper sense for a word but also provides the probabilistic evaluation for the identification confidence at the same time. A novel Instance Knowledge Network (IKN) is built to generate and maintain semantic knowledge at the word, type synonym set and instance levels. Related algorithms based on graph matching are developed to train IKN with probabilistic knowledge and to use IKN for probabilistic word sense disambiguation. Based on the Senseval-3 all-words task, we run extensive experiments to show the performance enhancements in different precision ranges and the rationality of probabilistic based automatic confidence evaluation of disambiguation. We combine our WSD algorithm with five best WSD algorithms in senseval-3 all words tasks. The results show that the combined algorithms all outperform the corresponding algorithms.

Keywords: natural language processing, word sense disambiguation

1 Introduction

Word sense disambiguation (WSD) is to identify the proper sense of words in the context. As a typical topic of natural language processing, WSD is widely used in machine translation, knowledge acquisition, information retrieval, etc. (Navigli 2009)

As a knowledge system in nature, WSD heavily relies on knowledge resources. Supervised WSD approaches mostly require manual sense tagged corpus. They provide the best performances in public evaluation (Palmer et al. 2001; Snyder and Palmer 2004). There are some knowledge based WSD systems which are built on a lexical knowledge base. They exploit the semantic relationships between concepts in semantic networks and computational lexicons (Yarowsky and Florian 2002; Cuadros and Rigau 2006).

Recently, a few graph based approaches for knowledge based WSD were proposed (Navigli and Velardi 2005; Sinha and Mihalcea 2007; Navigli and Lapata, 2007; Agirre and Soroa 2009). All these approaches are built at type level and the semantic relations between types.

Besides semantic relations, syntactic structures and relations are also valuable to WSD. Martinez et al. proposed a syntactic feature based WSD approach (Martinez et al. 2002). Fernandez-Amoros also presented a syntactic pattern WSD algorithm (Fernandez-Amoros 2004).

However there is no knowledge base for WSD systems which properly keeps both semantic relations and syntactic features in the context. Actually, relationships between two synsets may be different within different syntactic structures of the contexts. To reflect this difference, we consider keeping context related relationships between synsets in patterns at instance level.

Instance based learning (Ng and Lee 1996; Daelemans et al. 1999) is a promising approach for WSD. Instance based WSD algorithms do not neglect exceptions and accumulate further aid for disambiguation through new examples. However existing instance based WSD approaches do not consider the syntactic features. We believe that keeping syntactic structures as instance patterns will benefit WSD and it is a key point to combine semantic relationships and syntactic features.

Besides the above considerations about a knowledge base, we also concern about the accuracy of WSD results. The poor accuracy of WSD results is a bottleneck for the application of WSD (Navigli 2009). Inspired by the human learning process, we reckon that the confidence evaluation of WSD is important. Supposing a school girl is reading a text, even though she cannot understand the whole text, she knows which part she can understand and which part she cannot. Therefore, she can learn knowledge from the understood part with high confidence. The confidence is based on the evaluation on accuracy of understanding.

Martinez's approach (Martinez et al. 2002) provides high precision WSD results. This work emphasizes the importance of high precision WSD. However, it employs fixed rule-related thresholds without quantitative evaluation of disambiguation results. Preiss (Preiss 2004) proposed a probabilistic WSD approach. However, they convert the probabilistic results into qualitative ones without quantitative analysis and do not show how their probabilistic results relate to the accuracy of disambiguation.

In this paper, we propose a novel multilayer instance knowledge network (IKN) together with related probabilistic training and WSD algorithms. The IKN WSD algorithm combines the semantic relations and syntactic features together to provide WSD results with quantitative confidence evaluation.

The rest of the paper is structured as follows. In Section 2, we introduce IKN, its graph matching algorithm and the probabilistic training algorithm of IKN based on the graph matching algorithm. The IKN WSD algorithm is presented in Section 3. Section 4 presents the experiment results of the IKN WSD algorithm and its combinations with existing WSD algorithms. Related works are discussed in Section 5. Concluding remarks are given in Section 6.

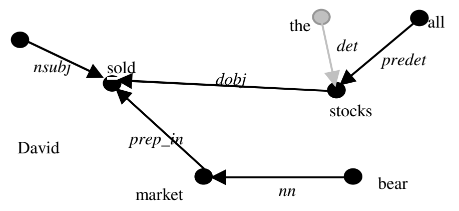

Figure 1 Semantic Dependency Graph for the sentence "David sold all the stocks in bear market."

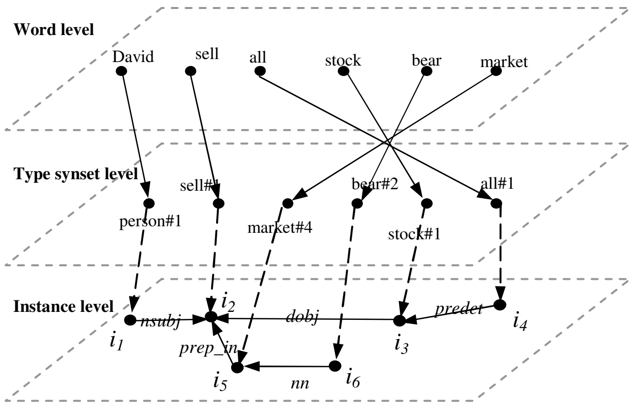

Figure 2 Connecting instance graph pattern to WordNet

2 Instance Knowledge Network (IKN)

Text understanding often requires contextual knowledge beyond the literal information of the text itself. Such contextual knowledge can be searched from the previously understood contents that are similar to the current text. Technically, such knowledge can be maintained in a semantic knowledge network which stores instances of word senses and relationships between these instances.

We propose a novel IKN - instance knowledge network to keep the contextual knowledge between word senses. IKN keeps the relationships between word senses not only at the type level (i.e., relations between type synsets) but also at the instance level (i.e., through a series of instance dependency graphs that are connected to the type synsets). It differs from most knowledge networks (Cuadros and Rigau 2008; Agirre and Soroa 2009) which keep the knowledge in the type synsets and their relations only.

2.1 Constructing IKN

We create IKN by extending WordNet (Fellbaum 1998) with a sense tagged corpus, such as, SemCor (Miller et al. 1993).

Figure 2 shows the simplified structure of IKN. The word level and the type synset level are inherited from WordNet. The instance level consists of a collection of *instance graph patterns* (*IGPs*), which are built from the texts in an existing corpus. We used SemCor in our system. The relationships between the instance level and the type synset level also need to be created. The brief procedure is as follows:

First, all the texts in the corpus are parsed into semantic dependency graphs using Stanford Dependency Parser (Stanford_Parser, Klein et al. 2002). Figure 1 shows a dependency graph for the sentence "David sold all the stocks in bear market." We suppose the sentence is sense tagged by WordNet3.0 synonym set (synset) , where David, sold, all, stocks, bear and market are assigned with synsets person#1, sell#1, all#1, stock#1, bear#2 and market#4, respectively. Each dependency graph is then inserted into IKN as an IGP by setting a unique identifier to each word node of the graph and the word node becomes an instance node of the IGP. Obviously, an IGP inherits the dependency relations between instance nodes from the dependency graph.

Then we connect instance nodes in each IGP at the instance level to type synsets at the synset level. Because each word in dependency graphs is sense tagged, each instance node has a sense tag inherits from the word. We connect each instance node to the type synset labeled by the sense tag. It is worthwhile noting that the relation between type synsets and instance nodes is one-to-many, i.e., a type synset may connect to multiple instance nodes.

In Figure 2, the IGP ($\{i_1, i_2, i_3, i_4, i_5, i_6\}$, $\{(i_1, i_2), (i_2, i_3), (i_2, i_5), (i_3, i_4), (i_5, i_6)\}$) at the instance level is obtained from the dependency graph in Figure 1. Instance node i_1 coming from word node "David" in Figure 1 is connected to its tagged sense synset person#1 at the synset level. To simplify the representation, the relations between type synsets are not given. IKN will be trained to obtain probabilistic knowledge (see Section 2.3).

2.2 Graph Matching Algorithm

Given a candidate sentence represented as a dependency graph *G* shown at the candidate level of Figure 3, we propose a graph matching algorithm to find all matching sub-graphs of IGPs in IKN for dependency graph *G*. The training algorithm for IKN and WSD algorithm will be based on this algorithm.

The algorithm can be described as 2 main steps.

(1). For each candidate word of dependency graph *G* at the candidate level, we find all instance nodes at the instance level that are *semantically related* to the word through IKN. We call these instance nodes as *semantic related instance nodes* (*SRINs*) of the candidate word.

(2). Among all SRINs of those candidate words in *G*, we discover all sub-graphs of IGPs that match *G* maximally.

Figure 3 shows a general picture on how the graph matching works. We now describe in detail these two main steps in the following two sub sections.

2.2.1 Finding Semantic Related Instance Nodes

Given candidate word *w* of candidate dependency graph *G* at the candidate level, we need to find all SRINs of *w* at the instance level through the word and type synset levels of IKN. This can be done in the following three sub steps.

Firstly, given a candidate word w*, we find all sense synsets of* w. We first find a unique symbol word *w*' at the word level for *w*. Then, we find the sense synsets of *w*' at the type synset level. Note that multiple sense synsets may exist for *w*'. We also consider these synsets as the sense synsets of *w*. For example in Figure 3, candidate word *market* in dependency graph G_1 has a symbol word *market* at the word level. Two synsets market#3 and market#4 at the type synset level are sense synsets of symbol word *market* as well as the sense synsets of candidate word *market* in G_1. Similarly, clear#16 is a sense synset of candidate word *cleared* in G_1, and assets#1 is a sense synset of candidate word *assets* in G_2.

Secondly, given a set of sense synsets of w*, we find all synsets that are semantically related to these sense synsets within the type synset level*. For each sense synset *s* of *w*, there may exist other synsets at the type synset level which are *semantically related* to *s*. We call these synsets as *semantically related synsets* (*SRSs*) of *s* in IKN. An SRS s_i of *s* is defined as a synset which holds one of the following three relationships with *s*.

(1). A single semantic relation exists from *s* to s_i within the type synset level;

(2). A semantic path exists from *s* to s_i within the type synset level, each step of the path has the same semantic relation and direction, and the semantic

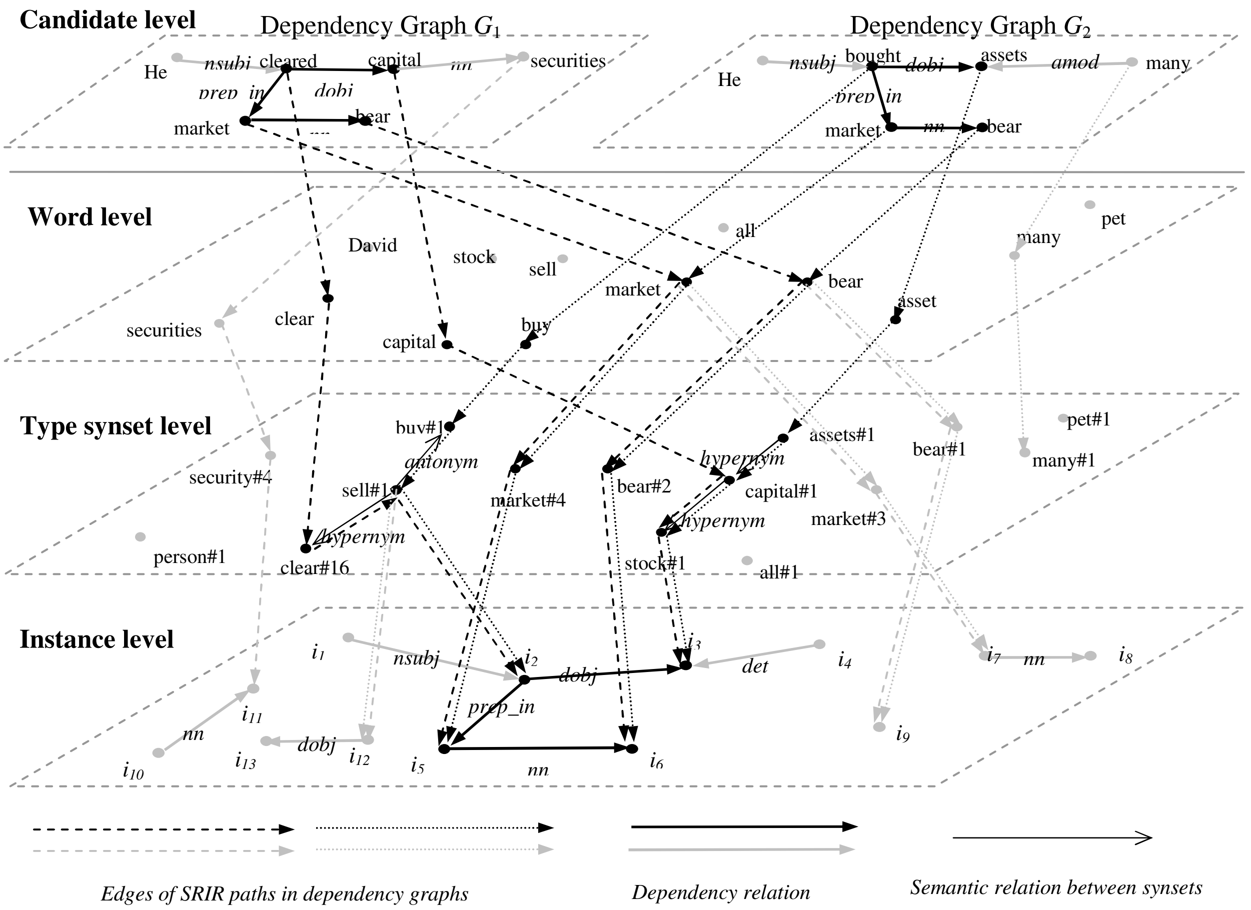

Figure 3 Graph Matching Algorithm on Instance Knowledge Network

relation is transitive;

(3). Sense synset s is an SRS of itself, denoted as *self*.

An SRS of a candidate word is an SRS of one of the sense synsets of the candidate word. For example, in Figure 3, the synset sell#1 is an SRS of candidate word *cleared* with the semantic relation *hypernym* to its sense synset clear#16; it is also an SRS of candidate word *bought* with the semantic relation *antonym* to its sense synset buy#1. The synset stock#1 is an SRS of *capital* with the direct *hypernym* relation to capital#1; it is also an SRS of *assets* to assets#1 with *hypernym* as an indirect transitive relation. We also consider market#3 and market#4 as SRSs of candidate word *market* because both market#3 and market#4 are sense synsets of *market* and they are deemed as SRSs with *self* semantic relation to the sense synsets.

Thirdly, *given a set of SRSs of* w*, we find all SRINs of these SRSs*. In IKN, a synset can have multiple instance nodes, so there can be multiple SRINs which are instances of each SRS of w. For example in Figire 3, sell#1 is an SRS of *cleared*, and it has two SRINs i_2 and i_{12}.

So when given a candidate word w, multiple SRINs may be returned. For each returned SRIN n, we define SRIR(w, n) as the *semantically related instance relationship* (*SRIR*) between w and n. From the above three sub steps, we know that such an SRIR stands for a path from w to n, including relations between a candidate word at the candidate level and its word symbol at the word level, between a word symbol and a sense synset at the type synset level, between a sense synset and an SRS both at the type synset level, and between an SRS and an instance node at the instance level. SRIR(w, n) can be denoted as (w, s, t, n) where w is the candidate word; s is the sense synset of w on the path; t is the semantic relation between s and the SRS s' on the path which directly connects to the instance node n. Given a particular SRIR r, we define the function SS(r) to return sense synset s and define the function SRC(r) to return the semantic relation t.

For example, in Figure 3, we denote SRIR(cleared, i_2) between the candidate word *cleared* and its SRIN i_2 as (cleared, clear#16, *hyponym*, i_2) and we can get SS(SRIR(cleared, i_2)) = clear#16 and SRC(SRIR(cleared, i_2)) = *hyponym.* Similarly we have SRIR(assets, i_3) as (assets, assets#1, *hypernym*, i_3), SS(SRIR(assets, i_3)) = assets#1 and SRC(SRIR(assets, i_3)) = *hyponym*; SRIR(capital, i_3) as (capital, capital#1, *hypernym*, i_3), SS(SRIR (capital, i_3)) = capital#1 and SRC(SRIR(capital, i_3)) = *hyponym*; SRIR(market, i_5) as (market, market#4, *self*, i_5), SS(SRIR(market, i_5)) = market#4 and SRC(SRIR(market, i_5)) = *self*; SRIR(bear, i_6) as (bear, bear#2, *self*, i_6), SS(SRIR bear, i_6)) = bear#2 and SRC(SRIR(bear, i_6)) = *self.*

It is worth mentioning that an SRS can also correspond to multiple sense synsets, and hence multiple candidate words. Consequently an instance node or SRIN can correspond to multiple sense synsets as well as multiple candidate words. For example in Figure 3, i_2 is an instance of sell#1, sell#1 is an SRS of candidate word *cleared* with the semantic relation *hypernym,* so we consider i_2 as an SRIN of candidate word *cleared* with semantic relation *hypernym* to its sense synset clear#16. Similarly, i_2 is also an SRIN of *bought* with semantic relation *antonym.*

2.2.2 Discovering Instance Matching Sub-graphs

When all SRINs of words in candidate dependency graph G are found, we now discover all sub-graphs of IGPs at the instance level that match G maximally. This can be achieved by a breadth-first traversal of G. To give a clear explanation, we divide it in following two steps.

Firstly, for each edge $e(w_1, w_2)$ being traversed in G, we find its matching edges in all IGPs at the instance level. An edge $e'(iw_1, iw_2)$ in an IGP G' is called a matching edge of $e(w_1, w_2)$ if it satisfies the following conditions:

(1). iw_1 and iw_2 are an SRIN of w_1 and w_2, respectively;

(2). The dependency relation d between w_1 and w_2 in G is the same as the dependency relation d' between iw_1 and iw_2 in the IGP.

For example in Figure 3, dependency relation between i_2 and i_3 is *dobj* which is the same as the dependency relation between the candidate words *cleared* and *capital.* Furthermore, there is an SRIR between *cleared* and i_2, and another SRIR between *capital* and i_3. Thus, we consider (i_2, i_3) as a matching edge of (cleared, capital) of G. Similarly, (i_2, i_5) is a matching edge of (cleared, market) with common dependency relation *prep_in*, and (i_5, i_6) is a matching edge of (market, bear) with common dependency relation *nn*.

Secondly, we try to connect the found matching edge $e'(iw_1, iw_2)$ to those previously found set of matching edges or sub-graphs in the IGP G'. We denote the set of previously found set of matching sub-graphs as S and S = { } at the beginning. $e'(iw_1, iw_2)$ can be connected to one sub-graph G_s in S if G_s includes a node iw that matches either iw_1 or iw_2, and corresponds to same candidate word w_1 or w_2. If none of such G_s exists in S, we simply add $e'(iw_1, iw_2)$ as a sub-graph to S.

When the traversal of G is done, we select the maximum sub-graph from S as the matching sub-graph of candidate dependency graph G from IGP G'. We call this sub-graph as an *instance matching sub-graph* (*IMSG*) of G.

For example in Figure 3, we start breadth-first traversal of G_1 from word *cleared* and S = { } at the beginning for one of the IGPs G' at the instance level. After (cleared, capital) is traversed, $S = \{\{(i_2, i_3)\}\}$ (for simplicity, only edges are recorded). After (cleared, market) is traversed, $S = \{\{(i_2, i_3), (i_2, i_5)\}\}$. After (market, bear) is traversed, $S = \{\{(i_2, i_3), (i_2, i_5), (i_5, i_6)\}\}$. After the traversal of G_1 is done, suppose there is no change to S, then the only sub-graph $\{(i_2, i_3), (i_2, i_5), (i_5, i_6)\}$ of S is the IMSG of G from IGP G'. There may be IMSGs from other IGPs for G.

2.3 Probabilistic Knowledge Training

Based on the graph matching algorithm, we train IKN by a sense tagged corpus. In our work, we use the SemCor to train IKN which was initially built from SemCor. From the training, probabilistic knowledge are obtained and attached to the IGPs.

In the training process, at first we parse the sense tagged corpus into candidate dependency graphs. Then we employ the graph matching algorithm to find IMSGs at the instance level of IKN. Each candidate dependency graph may match with many IMSGs. Each instance node pair in an IGP may be matched by many candidate dependency graphs in different IMSGs. Finally, we generate the conditional probabilities for each instance node pair in the IMSGs. Each time a pair of instance nodes is matched in an IMSG of a candidate dependency graph, some of the conditional probabilities related to them are generated or updated. In the following, we focus on the discussion on how we generate the conditional probabilities for a pair of instance nodes in an IGP of IKN.

For a pair of instance nodes i_1 and i_2 (denoted as $<i_1, i_2>$) in an IGP of IKN, we define two sets of conditional probabilities: $\mathrm{PI}(i_1, i_2)$ from i_1 to i_2, and $\mathrm{PI}(i_2, i_1)$ from i_2 to i_1. i_1 and i_2 may be directly or indirectly connected in the IGP. Each conditional probability in a set is created for its real WSD use and represents the *category* of sense synset pairs based on *a particular SRC pair* (refer to Section 2.2.1 for SRC definition) between $<i_1, i_2>$ and its matching word pairs. Through the training process, $\mathrm{PI}(i_1, i_2)$ and $\mathrm{PI}(i_2, i_1)$ are obtained and attached to $<i_1, i_2>$ as part of IKN. Due to space limitation, we only explain how we define a set of conditional probabilities in $\mathrm{PI}(i_1, i_2)$ as those in $\mathrm{PI}(i_2, i_1)$ can be defined similarly.

At first, for instance node pair $<i_1, i_2>$, we define condition C1 for that both i_1 and i_2 are in an IMSG of a candidate dependency graph from the training set. We count the number of times in the training process that C1 is satisfied as $\mathrm{Count_{all}}(i_1, i_2)$.

Each time $<i_1, i_2>$ satisfies C1 in a matching, a candidate word pair $<w_1, w_2>$ can be found in an candidate dependency graph and then the pair of SRIRs $\mathrm{SRIR}(w_1, i_1)$ and $\mathrm{SRIR}(w_2, i_2)$ are determined. Then we can define condition C2 for satisfying $\mathrm{SRC}(\mathrm{SRIR}(w_1, i_1)) = t_1$ and $\mathrm{SRC}(\mathrm{SRIR}(w_2, i_2)) = t_2$. We count the number of times in training process that both conditions C1 and C2 are satisfied as $\mathrm{Count_c}(i_1, t_1, i_2, t_2)$. Here, for the instance node pair $<i_1, i_2>$, we attempt to categorize the matching candidate word pairs based on different SRCs of the corresponding pairs of SRIRs. For each particular pair of SRCs $<t_1, t_2>$, we count the number of all matching candidate word pairs in the training set which satisfy the SRCs of matching. It is easy to see $\mathrm{Count_{all}}(i_1, i_2) = \sum \mathrm{Count_c}(i_1, t_i, i_2, t_j)$ for all $<t_i, t_j>$ pairs.

After that, for each candidate word w, we define the proper sense synset as $\mathrm{PSS}(w)$ which is the tagged sense synset for w. We define condition $\mathrm{C3}(i_1)$ satisfying $\mathrm{SS}(\mathrm{SRIR}(w_1, i_1)) = \mathrm{PSS}(w_1)$, i.e., the sense synset of the SRIR between w_1 and i_1 is the proper tagged sense of w_1. Similarly we define condition $\mathrm{C3}(i_2)$ satisfying $\mathrm{SS}(\mathrm{SRIR}(w_2, i_2)) = \mathrm{PSS}(w_2)$. We count the number of candidate word pairs in the training process that satisfy conditions C1, C2 and $\mathrm{C3}(i_1)$ as $\mathrm{Count_c}(i_1, t_1, \mathit{true}, i_2, t_2, \mathit{null})$. It stands for the size of the set of candidate word pairs, each pair $<w_1, w_2>$ in the set matches $<i_1, i_2>$, and the pair of SRCs is $<t_1, t_2>$, and the sense synset in $\mathrm{SRIR}(w_1, i_1)$ is the proper sense of w_1. Similarly, we count the number of candidate word pairs in the training process that satisfy conditions C1, C2 and *not* $\mathrm{C3}(i_1)$ as $\mathrm{Count_c}(i_1, t_1, \mathit{false}, i_2, t_2, \mathit{null})$. We also count the number of candidate word pairs in training process that satisfy conditions C1, C2, $\mathrm{C3}(i_1)$ and $\mathrm{C3}(i_2)$ as $\mathrm{Count_c}(i_1, t_1, \mathit{true}, i_2, t_2, \mathit{true})$. It stands for the size of the set of candidate word pairs, each pair $<w_1, w_2>$ in the set matches $<i_1, i_2>$, and the pair of SRCs is $<t_1, t_2>$, and the sense synsets in $\mathrm{SRIR}(w_1, i_1)$ and $\mathrm{SRIR}(w_2, i_2)$ are the proper senses of w_1 and w_2, respectively. Similarly, we count the number of candidate word pairs in training process that satisfy conditions C1, C2, $\mathrm{C3}(i_1)$ and *not* $\mathrm{C3}(i_2)$ as $\mathrm{Count_c}(i_1, t_1, \mathit{true}, i_2, t_2, \mathit{false})$.

It is not difficult to understand that $\mathrm{Count_c}(i_1, t_1, \mathit{true}, i_2, t_2, \mathit{false}) + \mathrm{Count_c}(i_1, t_1, \mathit{true}, i_2, t_2, \mathit{true}) = \mathrm{Count_c}(i_1, t_1, \mathit{true}, i_2, t_2, \mathit{null})$, and $\mathrm{Count_c}(i_1, t_1, \mathit{true}, i_2, t_2, \mathit{null}) + \mathrm{Count_c}(i_1, t_1, \mathit{false}, i_2, t_2, \mathit{null}) = \mathrm{Count_c}(i_1, t_1, \mathit{null}, i_2, t_2, \mathit{null}) = \mathrm{Count_c}(i_1, t_1, i_2, t_2)$.

Now, for each *particular SRC pair* $<t_1, t_2>$, we can define a conditional probability from i_1 to i_2 as $\mathrm{P}(i_2, t_2 \mid i_1, t_1) = \mathrm{Count_c}(i_1, t_1, \mathit{true}, i_2, t_2, \mathit{true})/\mathrm{Count_c}(i_1, t_1, \mathit{true}, i_2, t_2, \mathit{null})$. We give the formal interpretation of $\mathrm{P}(i_2, t_2 \mid i_1, t_1)$ as follows.

For all matched pairs $\{< w_{1i}, w_{2j} >\}$ of $<i_1, i_2>$ with $\mathrm{SRC}(\mathrm{SRIR}(w_{1i}, i_1)) = t_1$ and $\mathrm{SS}(\mathrm{SRIR}(w_{2j}, i_2)) = t_2$, let $s_{1i} = \mathrm{SS}(\mathrm{SRIR}(w_{1i}, i_1))$ and $s_{2j} = \mathrm{SS}(\mathrm{SRIR}(w_{2j}, i_2))$, $\mathrm{P}(i_2, t_2 \mid i_1, t_1)$ is the probability of s_{2j} being the proper sense of w_{2j} when s_{1i} is the proper sense of w_{1i}. $\mathrm{P}(i_2, t_2 \mid i_1, t_1)$ is added in set $\mathrm{PI}(i_1, i_2)$ to represent the category of those sense synset pairs based on $<t_1, t_2>$.

The conditional probabilities obtained will be attached to instance node pairs within IGPs. These conditional probabilities in different IGPs materialize the relationships of synset pairs in different contexts. The syntactic structures are also kept in IGPs. So the semantic relations and the syntactic structures can work together in our IKN WSD.

3 IKN WSD Algorithm

Based on IKN and the probabilistic knowledge obtained and attached to the instance level of IKN during the above training process, we propose a quantitative WSD approach, which comprises a word sense based iterative process and a probabilistic reasoning algorithm.

3.1 Word Sense Based Iterative Reasoning

We believe that the understanding of a word may depend on the understanding of other words in the context. As such, we propose an iterative reasoning process which is described as follows:

(a) We parse the text to be disambiguated into candidate dependency graphs and set an initial probability for every sense of the words in text. All the probabilities of senses for a particular word add up to 1.0.
(b) Apply our probabilistic reasoning algorithm to update the probability of each word sense by the related sense probabilities of other words. The probabilistic reasoning algorithm will ensure that the sense probabilities of each particular word add up to 1.0.
(c) Repeat (b) until the probabilities of word senses get stabilized.

Agirre and Soroa (Agirre and Soroa, 2009) proposed an Personalized PageRank WSD approach which is also an iteration algorithm. Their algorithm is based on a ranking system between type synsets. Different from their ranking apporach, the input and output of (b) in our algorithm are probability based, so we consider it as a reasoning algorithm.

In this paper, the initial probabilities are derived from WordNet. For each candidate word which has tag of part of speech (PoS) by Stanford parser, we find all the senses for this word in WordNet with the same PoS. WordNet provides *tagcount* between sense and word which shows the frequency of a sense for a word in a large scale corpus. The probability of a sense for a word is its *tagcount* (in WordNet) divided by the summation of the *tagcounts* of all senses of the word with the same PoS.

The probabilistic reasoning algorithm in (b) will be described in next 3 sub-sections. For each sense s of candidate word w in a candidate dependency graph, we find the set of maximum conditional probabilities $\{P(s \mid w, s_{ij}, w_i)\}$, where w_i is any surrounding word of w in the dependency graph and s_{ij} is any sense synset of w_i (to be discussed in Section 3.2). After that, we calculate the un-normalized sense probability $P_b^k(s|w)$ for the current step by combining the set of sense probabilities at the previous step $\{P^{k-1}(s_{ij}|w_i)\}$ and the set of corresponding maximum conditional probabilities $\{P(s \mid w, s_{ij}, w_i)\}$ (to be discussed in Section 3.3). Finally, we normalize $P_b^k(s|w)$ to $P^k(s|w)$ to ensure that the sense probabilities of each particular word add up to 1.0 and get the WSD results (to be discussed in Section 3.4).

3.2 Finding Maximum Conditional Probability

To a sense synset s of a candidate word w in a candidate dependency graph G and a sense synset s' of a surrounding candidate word w' of w, We attempt to find the maximum conditional probability $P(s \mid w, s', w')$ using a set of conditional probability sets. Each probability set $PI(i', i)$ in this set is selected because of a matched SRIN pair $<i', i>$ of the candidate word pair $<w', w>$. In the following, we explain how we get $P(s \mid w, s', w')$.

We first find all IMSGs of G in IKN by the graph matching algorithm. For each found IMSG G' which contains the matched SRIN pair $<i', i>$ of $<w', w>$, if $SS(SRIR(w', i')) = s'$ and $SS(SRIR(w, i)) = s$, then we search a conditional probability $P(i, t \mid i', t') \in PI(i', i)$ such that $t = SRC(SRIR(w, i))$ and $t' = SRC(SRIR(w', i'))$, here $<t', t>$ represents a particular category of sense synset pairs to which $<s', s>$ belongs. If such a probability exists, we add it into the probability set $PS(w', s', w, s)$ which is a set of conditional probabilities from s' to s. When we process all found IMSGs of G, $PS(w', s', w, s)$ records all relevant probabilities of matched SRIN pairs $\{<i', i>\}$ of the candidate word pair $<w', w>$. Now we set $P(s \mid w, s', w')$ as the maximum conditional probability among those probabilities in $PS(w', s', w, s)$ to represent the probability of s being the proper sense of w when s' is the proper sense of w'.

3.3 Combining Probabilities

To a sense synset s_p of candidate word w in a candidate dependency graph G, based on the conditional probabilities from the sense synsets of different surrounding candidate words in the context and the probabilities of these sense synsets from the previous step, we can calculate the probability of s_p for the current step.

At first, we calculate the weighted average of the conditional probabilities of sense s_p of w from a surrounding word w_i in the context as

$$P^k(s_p|w_i, w) = \sum_{j=1 \text{ to } x} P(s_p|w_i, s_{ij}, w)\, P^{k-1}(s_{ij}|w_i)$$

Where $P^{k-1}(s_{ij} \mid w_i)$ is the probability of sense s_{ij} of w_i at iteration step k-1 and x is the number of senses of w_i. $P^k(s_p \mid w_i, w)$ is the conditional probability of sense s_p of word w from word w_i in the context at iteration step k.

Then, we combine the conditional probabilities of sense s_p of w from multiple surrounding words according to Naive Bayes Approach. We define the probability of sense s_p of w as

$$P_b^k(s_p \mid w) = P_0(s_p \mid w) \prod_{w_i \in G} \frac{P^k(s_p \mid w_i, w)}{P_0(s_p \mid w)}$$

$P_0(s_p \mid w)$ is the start-up probability of the sense s_p. It is also the general probability of s_p for word w which is calculated from the *tagcount* of the senses of w in WordNet.

Finally, we normalize the probabilities to ensure that the probabilities of all senses of a word add up to 1. We calculate $P_w^k(w) = \sum P_b^k(s_p|w)$, $p = 1, 2, \ldots, y$ and y is the number of senses of candidate word w. Then we define the normalized probability

$$P^k(s_p|w) = \begin{cases} P_b^k(s_p|w) / P_w^k(w) & \text{if } P_w^k(w) > 0 \\ P_0(s_p|w) & \text{if } P_w^k(w) = 0 \end{cases}$$

3.4 Returning WSD Results with Confidences

At each iteration step, we choose the sense with the maximum normalized probability of a word as the result sense for the word and employ the probability of this sense as the *confidence* evaluation value for the word disambiguation. The confidence value is between 0 and 1. If there are two senses with the same probability, we select the one with smaller synset *rank* number of WordNet as the result sense of disambiguation. When disambiguation results remain unchanged for all the test words at an iteration step, we get the WSD result.

4 Experiments and Evaluation

In our experiments, IKN is created and then trained by the sense tagged corpus SemCor. All the experiment results are for senseval-3 (Mihalcea et al. 2004) all words task. The PoS tagging for our IKN WSD algorithm is based on Stanford dependency parser, so the results inherit its mistakes.

To get high precision WSD results, we define a threshold θ between 0 and 1 as the confidence value of our WSD algorithm. We choose the results with the confidence value equal to or higher than θ as the high precision WSD results.

In Section 3.1, for each sense synset, we presented the method for getting the start-up probabilities which are based on *tagcount* values in WordNet. To evaluate the experiment results, we present the baseline algorithm which takes the start-up probabilities for each sense synset of words, identifies the sense synset with the maximum probability as the result sense, and employs the probability of the result sense as the confidence value. If there are two sense synsets with the same probability, we also choose the one with lower *rank* number in WordNet.

Table 1 presents the precision and attempt coverage of our IKN WSD algorithm with different confidence thresholds θ, and *comparable* results of the baseline WSD algorithm on senseval-3 all words tasks. As our IKN WSD algorithm provides the result of those words with the *confidence* equal to or higher than θ, we define the number of attempted results as Attempt(θ) and the number of the correct results in these attempted results as Correct(θ). So the *precision* can be obtained by Correct(θ)/Attempt(θ). We define the total number of test words as n, then the *attempt coverage* is Attempt(θ)/n. In Senseval-3 all word tasks, $n = 2041$. The precision and attempt coverage are a pair of contradicting factors.

Normally, higher precision is associated with lower attempt coverage. We sort the WSD results of the baseline algorithm by confidence values in descending order. To compare fairly the precision of the baseline WSD results with our IKN WSD results for each particular θ, we select the baseline WSD results with biggest confidences. The number of the selected results is equal to Attempt(θ) (this implies that the same attempt coverage is selected for baseline as that of the IKN WSD). We define the correct results in these results as CorrectB(θ) to represent the comparable baseline WSD results. Then we can also get the precision for baseline results as CorrectB(θ)/Attempt(θ). Now we can compare the precisions between the IKN and the baseline WSD algorithms. As shown in Table 1, the precision of our IKN WSD results are higher than baseline WSD results in different attempt coverage ranges.

Table 1 High precision result comparison of IKN WSD algorithm and baseline WSD algorithm

Threshold θ	Precision			Attempt Coverage
	IKN	Baseline	Improvement	
0.9	89.6%	86.3%	3.3%	31.06%
0.8	86.9%	84.1%	2.8%	38.22%
0.7	84.1%	79.2%	4.9%	46.35%
0.6	75.9%	73.7%	2.2%	60.46%
0.5	71.2%	69.2%	2.0%	73.05%

The IKN high-precision WSD results can be combined with any existing WSD algorithm through the following method. For each test word w_i, we define $s_{\mathrm{IKN}}(w_i)$ as the result sense by IKN, $c_{\mathrm{IKN}}(w_i)$ as the confidence of disambiguation by IKN, $s_{\mathrm{E}}(w_i)$ as the result sense by the existing WSD algorithm, and θ as the confidence threshold for IKN WSD. Then we can get the result sense $s_{\mathrm{C}}(w_i)$ of w_i by the combined algorithm as

$$s_{\mathrm{C}}(w_i) = \begin{cases} s_{\mathrm{IKN}}(w_i) & \text{if } c_{\mathrm{IKN}}(w_i) \geq \theta \\ s_{\mathrm{E}}(w_i) & \text{if } c_{\mathrm{IKN}}(w_i) < \theta \end{cases}$$

Table 2 Recall of combined algorithms of IKN and existing algorithms in Senseval-3 all words tasks

	Single	Combine with IKN		
IKN Threshold θ	N/A	0.7	0.8	0.9
GAMBL	65.2%	65.2%	65.4%	65.4%
SenseLearner	64.6%	65.2%	65.1%	65.0%
Koc	64.1%	64.7%	64.7%	64.5%
R2D2	62.6%	63.8%	63.4%	63.2%
Meaning-allwords	62.4%	63.6%	63.6%	63.5%

Because the confidence value $c_{\mathrm{IKN}}(w_i)$ of each word w_i is generated in our IKN WSD algorithm, the decision of result selection based on $c_{\mathrm{IKN}}(w_i)$ is also made by the algorithm. So it is reasonable to consider the combined results as the results of the new combined algorithm. If our IKN WSD algorithm selects the results based on the imprecise confidence values, there is no guarantee that the combined result is always better than that generated by the existing algorithm.

Table 2 shows the performance of combined algorithms of IKN and the top five algorithms (Snyder and Palmer, 2004) in Senseval-3 all words tasks. Since the combined algorithms work for the full attempt coverage, we compare them with the existing algorithms by *recall*. When the confidence threshold reaches 0.8, each of the combined results is better than that of the corresponding single algorithm. This shows that IKN has better performance than existing algorithms in the set of test words with high disambiguation confidence. These results also show that high precision WSD methods could be used to improve the performance of existing algorithms.

5 Related Work and Discussion

Personalizing PageRank algorithm (Agirre and Soroa, 2009) is an iterative ranking process between word senses in the context. The basic principle is similar to IKN. This ranking algorithm is based on the semantic distance between concepts in a lexical knowledge base (LKB). However, their LKB is a type level network. The shortest semantic distance between two concepts has been fixed when the LKB was built. Although they extract sub-graphs for given input context, the shortest semantic distance between concepts is not changed. In other words, the shortest semantic distance is context free. The algorithms proposed by Mihalcea and Sinha (Mihalcea 2005; Sinha and Mihalcea 2007) are built on similarities between senses, which are also context free. SSI WSD algorithm (Navigli and Velardi 2005) is proposed on the basis of graph pattern matching. However, their graph patterns do not contain the context sensitive syntactic feature either. Essentially, the graph matching between structural specifications of concepts is a measure of similarity between concepts too. Because the structural specifications are fixed for the concepts, the best matching structural specification is context free as well.

In IKN, we employ the maximum conditional probability of a sense synset from other sense synsets. Its effect is similar to the shortest semantic distance and the similarity in the above works. Different from LKB based algorithms and SSI, the conditional probabilities of IKN are kept at instance level. The maximum probability is determined by the graph matching between candidate graph and IGPs. Even containing the same two words, candidate graphs of different contexts may match different IGPs due to different syntactic structures. So the maximum conditional probability between sense synsets in IKN is syntactic context sensitive. This additional context relevance provides higher accuracy for WSD.

6 Conclusion and Future Works

In this paper, we have proposed a new instance knowledge network – IKN, and its graph matching algorithm. We have also developed the training algorithm for IKN and the probabilistic WSD algorithm using IKN. Our experimental study reveals reasonable performance of the IKN WSD algorithm. The high performance of combined algorithms of IKN with existing WSD algorithms shows that IKN based WSD can provide better results than the existing algorithms for the test words with high disambiguation confidence.

So far, we have conducted preliminary work on IKN based probabilistic WSD and there is plenty of room for improvement in the future. The dependency graphs we used at current stage are only for individual sentences due to the direct use of the output of the dependency parser. The IKN WSD algorithm will be beneficial from joining the sentence dependency graphs together by a high precision co-reference resolution algorithm. In addition, fuzzy graph matching may improve the attempt coverage. We will also study the semi-supervised learning algorithms on IKN.

7 References

E. Agirre and A. Soroa. 2009 *Personalizing PageRank for Word Sense Disambiguation.* In EACL 2009

M. Cuadros and G. Rigau. 2006. *Quality assessment of large scale knowledge resources.* In EMNLP 2006.

M. Cuadros and G. Rigau. 2008. *KnowNet: Building a Large Net of Knowledge from the Web.* In COLING 2008.

W. Daelemans, A. Van Den Bosch and J. Zavrel. 1999. *Forgetting exceptions is harmful in language learning.* Mach. Learn. 34(1) 1999.

C. Fellbaum. 1998. *WordNet – an electronic lexical database.* MIT Press, 1998.

D. Fernandez-Amoros. 2004. *WSD Based on Mutual Information and Syntactic Patterns* In ACL Senseval-3 Workshop 2004.

D. Klein and C. Manning. 2002. *Fast Exact Inference with a Factored Model for Natural Language Parsing.* In NIPS 2002.

D. Martinez, E. Agirre and L. Marquez. 2002. *Syntactic features for high precision word sense disambiguation.* In COLING 2002,.

R. Mihalcea. 2005. *Unsupervised large-vocabulary word sense disambiguation with graph-based algorithms for sequence data labeling.* In HLT2005

R. Mihalcea and P. Edmonds, Eds. 2004. In ACL Senseval-3 Workshop 2004.

H. Miller, C. Leacock, R. Tengi and R. Bunker. 1993. *A semantic concordance.* In ARPA Workshop on HLT, 1993.

R. Navigli. 2009 *Word sense disambiguation: A survey.* ACM Computing Surveys, 41(2), 2009.

R. Navigli and M. Lapata. 2007. *Graph connectivity measures for unsupervised word sense disambiguation.* In IJCAI 2007.

R. Navigli and P. Velardi. 2005. *Structural Semantic Interconnections: A Knowledge-Based Approach to Word Sense Disambiguation.* IEEE Trans. Pattern Anal. Mach. Intell, 27(7) 2005.

H. Ng and H. Lee, 1996. *Integrating multiple knowledge sources to disambiguate word senses: An examplar-based approach.* In ACL 1996.

M. Palmer, C. Fellbaum, S. Cotton, L. Delfs, and H.T. Dang. 2001. *English tasks: All-words and verb lexical sample.* In SENSEVAL-2 Workshop 2001.

J. Preiss. 2004. *Probabilistic word sense disambiguation.* Journal of Computer Speech and Language, 18(3) 2004.

R. Sinha and R. Mihalcea. 2007. *Unsupervised graphbased word sense disambiguation using measures of word semantic similarity.* In ICSC 2007.

B. Snyder and M. Palmer. 2004. *The English all-words task.* In ACL Senseval-3 Workshop 2004.

Stanford_Parser. http://nlp.stanford.edu/software/lex-parser.shtml

D. Yarowsky and R. Florian, 2002. *Evaluating sense disambiguation across diverse parameter spaces.* J. Nat. Lang. Eng. 9(4), 2002.