Exploring and Expanding the Use of Lexical Chains in Information Retrieval

Technical Report

Terry L. Ruas, William I. Grosky

University of Michigan – Dearborn, Computer and Information Sciences, Dearborn, USA
truas@umich.edu, wgrosky@umich.edu

Summary: *This technical report explains our advances in the arena of exploring lexical chains construction using WordNet and proposed algorithms for different types of structures.*

1. Best Synset Disambiguation

The Best Synset Disambiguation is a subroutine that applies and extends the concept of WSD, but considers the synsets extracted from w_i, w_{i-1} and w_{i+1}. WSD is the problem in which one must decide which *sense* is better suited for a word in a sentence, given that this word has multiple meanings and each one of them has an influence from other words. We explore this question through the BSID of a word w_i, which is selected by considering its predecessor (w_{i-1}), producing a synset called *FormerSynsetID*(w_i) (FSID(w_i)) and its successor (w_{i+1}) word, producing a synset called *LatterSynsetID*(w_i) (LSID(w_i)) as the algorithm in Fig. 1 illustrates.

```
for each word w_i in W where i={1,2,…,n}:
   apply WSD algorithm between <w_i,w_i+1> and <w_i,w_i-1> and get scores for all
synsets
   set <fs_i,fs_i-1> as the pair of synsets with the highest score, using <w_i,w_i-1>
   set <ls_i,ls_i+1> as the pair of synsets with the highest score, using <w_i,w_i+1>

   let FSID(w_i) = fs_i
   let LSID(w_i) = ls_i
return the pairs <FSID(w_i),LSID(w_i)> for i={1,2,…,n}
```

Fig. 1. Former and Latter Synset Selection.

The selection of FSID(w_i) and LSID(w_i) is based on the score provided by all possible combinations between all *senses* of the pairs (w_i,w_{i-1}) and (w_i,w_{i+1}), respectively. This score is calculated applying Wu & Palmer's algorithm [1, 2], defined in Eq. 1.

$$sim_{WP}(c_1, c_2) = \frac{2 * depth(lso(c_1,c_2))}{len(c_1,c_2)+2*depth(lso((c_1,c_2))} \quad (1)$$

Where $len(c_1,c_2)$ is the length of the shortest path from synset c_1 to synset c_2 in WN; $lso(c_1,c_2)$ is the lowest common subsumer of synset c_1 and synset c_2; $depth(c_1)$ is the length of the path to synset c_1 from the root *entity* (initial synset) in WN. More information about Wu & Palmer's algorithm can be found in [2], where they conduct a small survey about the most popular WSD algorithms available in the literature.

After each word w_i has its own FSID and LSID, it is necessary to find the BSID for this given word w_i. For this task, we introduce an algorithm called *Best Synset Disambiguation* (BSD) to suggest the BSID, using as parameters LSID and FSID. Three cases are considered prior to its selection: (a) if FSID(w_i) and LSID(w_i) are equal, then BSID(w_i) = FSID(w_i) = LSID(w_i); (b) the lowest common subsumer between FSID(w_i) and LSID(w_i), given a threshold; and (c), if (b) produces an empty set, the deepest *synset* among FSID(w_i) and LSID(w_i) is chosen. In case both have the same depth, one is chosen randomly. It is important to mention that, as we traverse the graph in WN for the hypernyms extraction (for each FSID and LSID), we consider the first hypernym in each level, for each synset. Considering that WN organizes the elements in each synset from most to least frequent usage, and we are generalizing the concepts as we move towards the root, it is only natural that we extract a hyerpnym that will provide the most diffused element. Therefore, the first hypernym in every upper level will provide greater probability of an intersection

with another synset when we build our extended lexical chains. In case both have the same depth, one is chosen randomly. Fig. 2 shows in detail the BSD algorithm.

```
for each <FSID(wi),LSID(wi)> corresponding to word wi in W, where i={1,2,…,n}:
  if (FSID(wi) = LSID(wi)) then BSID(wi) = FSID(wi) = LSID(wi)
     else
       set hypernym cut-off for (FSID(wi),LSID(wi))
       extract the set HFSID(wi) of all hypernyms from FSID(wi)
       extract the set HLSID(wi) of all hypernyms from LSID(wi)
       let γ = the lowest common subsumer between HFSID(wi) and HLSID(wi) using a
          defined cut-off
       if (γ exists) then BSID(wi) = γ
          else
           depth1 = depth(HFSID(wi))
           depth2 = depth(HLSID(wi))
           if (depth1 = depth2) then BSID(wi) = Random(HFSID(wi),HLSID(wi))
              else BSID(wi) = arg max depth(x), for x = HFSID(wi) or x = HLSID(wi)
  return BSID(wi)
```

Fig. 2. Best Synset Disambiguation (BSD) Algorithm.

2. Flexible Lexical Chain Extraction

Once all words have their BSID selected, we start building our lexical chains in a two-phase subroutine called Lexical Synset Chain Extraction Module. To the best of our knowledge, this module is introducing two novel contributions. First (a), we construct parametrized flexible lexical chains, considering an adaptive structure of synsets based on multilevel hypernyms and second (b), we transform these flexible chains into fixed structures to better represent the semantic values extracted from these synsets.

In (a), we introduce an algorithm called *Flexible Lexical Chains* (FLC), which extracts these chains, evaluating if a new synset (of a word w_i), or its hypernyms, present lexical cohesion among themselves and the current chain in construction. If the evaluated synset has semantic affinity with the current chain (within a certain level of abstraction in WN) this new synset is incorporated to the chain. Otherwise, a new chain will be initialized to capture the next semantic representation.

First, we begin a new FLC inserting the first word, w_1, from the text into an initial chain structure. Call this first chain FLC(1) and let BSID(w_1) be the synset id representing the FLC. For each word, w_i, in the document, where $i = 2,3,..,n$, we verify if BSID(w_i) is equal to BSID(w_{i-1}). If they are the same, we just add w_i into the current chain. In case they are not the same, we need to investigate if there is any semantic relationship shared between BSID(w_i) and BSID(w_{i-1}). This is done by extracting all hypernyms from BSID(w_i) and BSID(w_{i-1}), called α and β respectively.

Next, we choose the *lowest common subsumer*, called γ, between α and β given a certain cut-off. This threshold is to avoid that the relatedness between α and β is too general, since all synsets in WN are connected by the root *entity*. If γ doesn't exist, it means that BSID(w_i) and BSID(w_{i-1}) are not semantically related in a non-trivial sense, so a new FLC must be initialized. In case γ does exist, w_i is included into the current FLC and the synset representing the current FLC is updated. Fig. 3 shows in detail each step in the FLC algorithm.

```
set NCh = 1
define FLC(NCh) as the initial flexible lexical chain, containing only BSID(w1)
set BSID(w1) as synset representative of FLC(NCh); FLCID(NCh) = BSID(w1)
for each BSID(wi) in D, where i={2,…,n}:
  if (BSID(wi) = BSID(wi-1)) then BSID(wi) is added to FLC(NCh)
    else
     extract the set α of all hypernyms of BSID(wi-1)
     extract the set β of all hypernyms of BSID(wi)
     let γ = lowest common subsumer between α and β using a defined cut-off
     if (γ doesn't exist) then
       set NCh = NCh+1
       define FLC(NCh) as the next flexible lexical chain, containing only
          BSID(wi) and reinitialize FLCID(NCh) to BSID(wi)
       else
         include BSID(wi) in FLC(NCh)
         set FLCID(NCh) = γ
return FLC(j) for j=1,…,NCh
```

Fig. 3. Flexible Lexical Chain (FLC) Algorithm.

3. Flexible to Fixed Lexical Chain Extraction (Flex2Fixed)

After all FLC are produced, we can convert these flexible chains in fixed structures (Flex2Fixed). We assign to all words, w_i, in the document, the FLCID of the FLC in which this word-synset appears.

For (b) above, after all pairs of words and synsets are arranged, we divide the document into chunks of size k with respect to the number of synsets, represented by cw_k. Each chunk corresponds to a fixed lexical chain. For each cw_k, we extract the dominant synset (the one that appears most often in the chunk), called θ, and assign it to represent cw_k. If there is more than one dominant synset, θ is chosen randomly. Fig. 4 shows in detail the Flex2Fixed algorithm, while Fig. 5 is a pictorial representation of the process itself. Thus, each fixed lexical chain is represented by a synset. The total number of synsets used in our overall document representation is the union over all documents of the synsets representing all the fixed lexical chains in that document. Call this value *NSynsets*. Let the synsets used in our document representation be syn_1, syn_2, …, $syn_{NSynsets}$.

```
for each word occurrence wi in D, where i={1,…,n}:
   set synset(wi) = p, where wi occurs in FLC(k)and p = FLCID(k)
split D into fixed-sized chunks of k words each
for each chunk cwj, where j={1,…,NChunks}:
   let wj,1,…,wj,k be the word occurrences in chunk cwj
   let θj = <synset(wj,1),synset(wj,2),…,synset(wj,k)>
   represent chunk cwj by the dominant synset of θj, dominant(cwj)
return dominant(cwj)for j=1,…,NChunks
```

Fig. 4. Flex2Fixed (F2F) Algorithm.

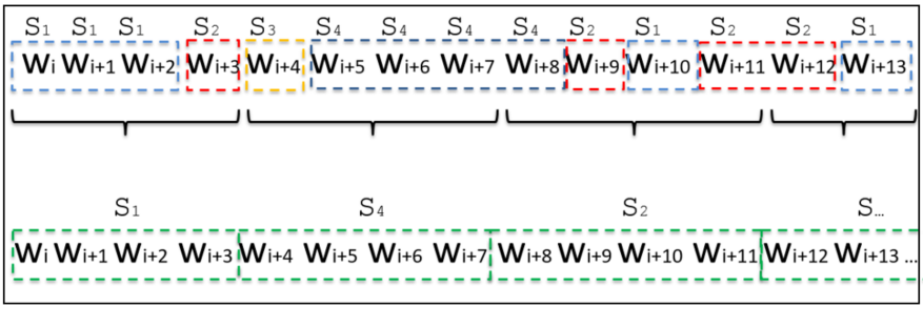

Fig. 5. Flex2Fixed Process.

4. Fixed Lexical Chain Extraction

Once all words have their BSID selected, we can also build fixed chains directly. To the best of our knowledge, there is no previous work in the available literature that constructs parametrized fixed lexical chains for multi-level of hypernyms and chain size.

In this section we introduce an algorithm called Fixed Lexical Chains (FXLC), which extracts these chains given a pre-defined number of *synset*s that these chains should contain. For each chain c_d, we extract all hypernyms (including the initial *synset*s) from all the *synset*s and call this set λ. Since each chain must be properly represented, only dominant (appearing in at least half of c_d) *synset*s, β, are considered. If there is no dominant *synset* in the chain, λ is selected instead. Let δ be either λ or β, depending on the preceding condition.

Next, we choose a subset of δ, called ε, of those *synset*s that are not too close to the root (entity) in WN. This is to prevent providing a *synset* too general for our chain. If ε is empty, all *synset*s in δ are considered instead. Let Ω be either ε or δ, depending on the preceding condition.

Then, from Ω we extract all maximally occurring *synset*s, α. We then construct γ, a subset of *synset*s that occur at the deepest level of α. If α has more than one *synset*, the *synset* to represent the initial chain c_d is then a random *synset* from α. Since we already limit the search in the hypernyms to guarantee a certain level of generality, now we want to maintain the semantic value within each fixed chain. See **Fig. 6** for details.

1. **select** the set $\lambda(c_d)$ of all hypernyms from each chain $\boldsymbol{c_d}$
2. **select** set β of all synsets that appear in at least half of $\boldsymbol{c_d}$
3. **if** $(\beta = \phi)$ **then** $\delta = \lambda(c_d)$ **else** $\delta = \beta$
4. **perform** cut-off in δ based on a chosen limit, producing ε
5. **if** $(\varepsilon = \phi)$ **then** $\Omega = \delta$ **else** $\Omega = \varepsilon$
6. **extract** the set α of all maximally occurring synsets in Ω
7. **select** the set γ of maximally deepest synsets in α
8. **return** a random synset from γ as representing the chain c_d

Fig. 6. Fixed Lexical Chains (FXLC) Algorithm.

Thus, each chain is represented by a *synset*. The total number of *synset*s used in our document representation is the union over all documents of the *synset*s representing all the fixed document chains. Call this value N*Synset*s. Let the *synset*s used in our document representation be syn_1, syn_2,…, $syn_{NSynsets}$.

Consider document *d*. For each $1 \leq i \leq$ N*Synset*s, we define *h(d,i)* to be the histogram of relative distances between consecutive occurrences of syn_i in document *d*. Note that the number of bins of *h(d,i)* and *h(e,j)* are the same for any 2 documents *d*, *e*, and *synset*s *i*, *j*. Also, for *h(d,i)*, if syn_i does not occur in document *d*, then the histogram consists of all 0's. Document *d* is then represented by the normalized concatenation of $h(d,syn_1),\ldots,h(d,syn_{NSynsets})$.

5. Sample Results of using Lexical Chains for Document Similarity

This section shows some preliminary results of applying Flexible and Fixed Lexical Chain Extraction in a corpus of 10 webpages from Wikipedia: 05 from dog categories and 05 from computer categories. Each document is used/compared against the entire corpus through the use of cosine similarity, where 1 means a perfect match and 0 the opposite.

Flex_BA	DOG A	DOG B	DOG C	DOG D	DOG E	COMP A	COMP B	COMP C	COMP D	COMP E
DOG A	1	0.5879	0.5932	0.6114	0.477	0.2928	0.0775	0.2118	0.1467	0.322
DOG B	0.5879	1	0.6825	0.7684	0.5343	0.2119	0.0731	0.1332	0.1582	0.1862
DOG C	0.5932	0.6825	1	0.6895	0.5118	0.3091	0.0945	0.1788	0.2272	0.3038
DOG D	0.6114	0.7684	0.6895	1	0.5575	0.1623	0.0518	0.109	0.1053	0.1514
DOG E	0.477	0.5343	0.5118	0.5575	1	0.2076	0.0564	0.158	0.1085	0.2275
COMP A	0.2928	0.2119	0.3091	0.1623	0.2076	1	0.3827	0.7446	0.6829	0.7553
COMP B	0.0775	0.0731	0.0945	0.0518	0.0564	0.3827	1	0.2754	0.6658	0.2273
COMP C	0.2118	0.1332	0.1788	0.109	0.158	0.7446	0.2754	1	0.4557	0.6547
COMP D	0.1467	0.1582	0.2272	0.1053	0.1085	0.6829	0.6658	0.4557	1	0.475
COMP E	0.322	0.1862	0.3038	0.1514	0.2275	0.7553	0.2273	0.6547	0.475	1

Fig. 7. FLC document similarity considering the relative distance of chains.

Flex_TA	DOG A	DOG B	DOG C	DOG D	DOG E	COMP A	COMP B	COMP C	COMP D	COMP E
DOG A	1	0.2971	0.3265	0.3377	0.3245	0.1933	0.0944	0.1561	0.1319	0.2265
DOG B	0.2971	1	0.2659	0.33	0.305	0.1253	0.0914	0.1431	0.1487	0.1217
DOG C	0.3265	0.2659	1	0.4186	0.3456	0.2151	0.0876	0.1411	0.1438	0.2356
DOG D	0.3377	0.33	0.4186	1	0.3361	0.1681	0.0859	0.1514	0.1032	0.1732
DOG E	0.3245	0.305	0.3456	0.3361	1	0.1751	0.0967	0.1498	0.1224	0.1975
COMP A	0.1933	0.1253	0.2151	0.1681	0.1751	1	0.2809	0.3624	0.3158	0.4622
COMP B	0.0944	0.0914	0.0876	0.0859	0.0967	0.2809	1	0.2178	0.3808	0.1865
COMP C	0.1561	0.1431	0.1411	0.1514	0.1498	0.3624	0.2178	1	0.2259	0.3375
COMP D	0.1319	0.1487	0.1438	0.1032	0.1224	0.3158	0.3808	0.2259	1	0.2392
COMP E	0.2265	0.1217	0.2356	0.1732	0.1975	0.4622	0.1865	0.3375	0.2392	1

Fig. 8. FLC document similarity considering the absolute position of chains.

FIXED Algorithm										
Fixed_BA	DOG A	DOG B	DOG C	DOG D	DOG E	COMP A	COMP B	COMP C	COMP D	COMP E
DOG A	1	0.6328	0.8718	0.7707	0.8244	0.5644	0.2876	0.2932	0.3438	0.488
DOG B	0.6328	1	0.719	0.7119	0.6457	0.3222	0.193	0.1374	0.2587	0.2342
DOG C	0.8718	0.719	1	0.8715	0.8308	0.6518	0.3488	0.337	0.4942	0.5267
DOG D	0.7707	0.7119	0.8715	1	0.8284	0.3772	0.1975	0.1912	0.2863	0.2949
DOG E	0.8244	0.6457	0.8308	0.8284	1	0.5147	0.2578	0.2876	0.3518	0.4282
COMP A	0.5644	0.3222	0.6518	0.3772	0.5147	1	0.4774	0.7281	0.6953	0.9093
COMP B	0.2876	0.193	0.3488	0.1975	0.2578	0.4774	1	0.3529	0.4712	0.4186
COMP C	0.2932	0.1374	0.337	0.1912	0.2876	0.7281	0.3529	1	0.3744	0.8445
COMP D	0.3438	0.2587	0.4942	0.2863	0.3518	0.6953	0.4712	0.3744	1	0.5194
COMP E	0.488	0.2342	0.5267	0.2949	0.4282	0.9093	0.4186	0.8445	0.5194	1

Fig. 9. FXLC document similarity considering the relative distance of chains.

Fixed_TA	DOG A	DOG B	DOG C	DOG D	DOG E	COMP A	COMP B	COMP C	COMP D	COMP E
DOG A	1	0.3243	0.4965	0.4484	0.5173	0.3659	0.2168	0.2613	0.2335	0.3374
DOG B	0.3243	1	0.3473	0.3626	0.356	0.2192	0.1579	0.155	0.1941	0.1907
DOG C	0.4965	0.3473	1	0.6274	0.5414	0.435	0.2671	0.2757	0.3004	0.4398
DOG D	0.4484	0.3626	0.6274	1	0.5167	0.3187	0.1827	0.2282	0.2354	0.2882
DOG E	0.5173	0.356	0.5414	0.5167	1	0.3532	0.2067	0.2383	0.2907	0.3034
COMP A	0.3659	0.2192	0.435	0.3187	0.3532	1	0.3705	0.5108	0.4603	0.6438
COMP B	0.2168	0.1579	0.2671	0.1827	0.2067	0.3705	1	0.3215	0.3586	0.3514
COMP C	0.2613	0.155	0.2757	0.2282	0.2383	0.5108	0.3215	1	0.3575	0.585
COMP D	0.2335	0.1941	0.3004	0.2354	0.2907	0.4603	0.3586	0.3575	1	0.383
COMP E	0.3374	0.1907	0.4398	0.2882	0.3034	0.6438	0.3514	0.585	0.383	1

Fig. 10. FXLC document similarity considering the absolute position of chains.

References

1. Wu, Z., Palmer, M.: Verb semantics and lexical selection. 32nd Annu. Meet. Assoc. Comput. Linguist. 133–138 (1994).
2. Meng, L., Huang, R., Gu, J.: A Review of Semantic Similarity Measures in WordNet. Int. J. Hybrid Inf. Technol. 6, 1–12 (2013).