Optimización del Algoritmo de WSD SSI-Dijkstra

Rubén Chihuailaf – Mauro Castillo
Depto. Informática y Computación
Universidad Tecnológica Metropolitana
Av. José Pedro Alessandri 1242,Santiago - CHILE
rubenchihuailaf@gmail.com, mauro.castillo@utem.cl

Gorka Blanco – Aritza Ledesma – German Rigau
Departamento LSI, Facultad de Informática
Universidad del País Vasco UPV-EHU
Paseo Manuel de Lardizábal, 1. E-20018, ESPAÑA
{a.ledesma003, gblanco002}@ikasle.ehu.es,
german.rigau@ehu.es

Resumen **- En este trabajo se presenta SSI-Dijkstra-Fast, una nueva versión optimizada del algoritmo SSI-Dijkstra para la resolución de la ambigüedad semántica de la palabras (en inglés Word Sense Disambiguation), un área de investigación muy relevante en el Procesamiento del Lenguaje Natural (PLN). En pruebas empíricas sobre un mismo corpus y máquina hemos demostrado que a) los resultados de SSI-Dijkstra-Fast son muy similares a los del SSI-Dijkstra original y b) que en ese mismo corpus SSI-Dijkstra-Fast realiza la misma tarea de desambiguación en menos de la mitad de tiempo. El código resultante está disponible bajo licencia GNU.**

Palabras claves – Inteligencia artificial, Procesamiento del lenguaje natural, desambiguación de sentidos de palabras.

I. Introducción

La *resolución de la ambigüedad semántica de las palabras* (WSD, del inglés Word Sense Disambiguation) es el proceso de identificación del sentido de una palabra polisémica cuando se utiliza en una oración o contexto. WSD es un área de investigación muy relevante en el Procesamiento del Lenguaje Natural (PLN). Libros genéricos de PLN dedican capítulos separados a la WSD (Manning & Schütze 1998, Jurafsky et al. 2000, Dale et al. 2000). También hay números especiales sobre WSD en revistas de PLN (Ide & Veronis 1998, Edmonds & Kilgarrif 2002), estados del arte (Navigli 2009), y libros que se centran en este tema (Ravin & Leacock 2000, Stevenson & Wilks 2003, Agirre & Edmonds 2006). Como se ha demostrado empíricamente en los últimos ejercicios de SensEval[1] y SemEval[2], a pesar de la amplia gama de enfoques investigados y el gran esfuerzo dedicado a resolver este problema, asignar el significado apropiado a las palabras en su contexto ha resistido todos los intentos de ser abordado con éxito.

Sin embargo, aunque los resultados distan de ser perfectos, WSD ha demostrado ser útil para mejorar tareas como en el análisis sintáctico (Syntactic Parsing) (Agirre et al. 2008), la recuperación de información (Information Retrieval) (Zhong & Ng 2012), la traducción automática (Machine Translation) (Carpuat & Wu 2007) o la extracción de la información (Information Extraction) (Chai & Biermann 1999).

Tradicionalmente, los sistemas de WSD han usado un enfoque superficial. Estos enfoques no tratan de comprender completamente el texto. Por lo general, sólo tienen en cuenta heurísticas simples para determinar el significado de una palabra en un contexto particular.

Últimamente, los métodos de WSD que usan bases de conocimiento representadas como grafos han atraido mucho la atención de la comunidad de PLN (Navigli & Velardi 2005, Mihalcea 2005, Sinha & Mihalcea 2007, Navigli & Lapata 2007, Cuadros & Rigau 2008, Agirre & Soroa 2009, Laparra et al. 2010). Estos métodos explotan distintas propiedades estructurales de los grafos que subyacen a una red semántica o base de conocimiento. En general, los métodos de WSD basados grafos exploran las interralaciones existentes en una base de conocimiento entre los sentidos de las palabras de un contexto dado.

Los métodos basados en grafos tienen grandes ventajas. En primer lugar, no requieren de entrenamiento. Además, estos métodos son independientes del idioma ya que sólo necesitan una base de conocimiento del idioma o enlaces multilíngües a un grafo genérico. Normalmente, los algoritmos de WSD basado en grafos usan WordNet (Fellbaum 1998) como base de conocimiento. Finalmente, los algoritmos basados en grafos también obtienen buenos resultados cuando se aplican a palabras estrechamente relacionadas entre sí (Cuadros & Rigau 2008, Laparra et al. 2010).

En el presente trabajo presentamos SSI-Dijkstra-Fast, una nueva versión mucho más eficiente del algoritmo de WSD basado en grafos SSI-Djikstra+ (Laparra et al. 2010).

Tras esta breve introducción, en la sección II presentaremos los conceptos necesarios para comprender las contribuciones de este trabajo. En la sección III presentaremos el algoritmo SSI-Dijkstra-Fast. A continuación, en la sección IV presentaremos el marco de evaluación y en la sección V los resultados experimentales obtenidos. Finalmente la sección IV concluirá este trabajo y mostrará líneas futuras de investigación.

II. Antecedentes

A. *WordNet como grafo*

WordNet[3] (Fellbaum 1998) es una base de conocimiento léxica para el idioma inglés.

1 http://www.senseval.org
2 http://en.wikipedia.org/wiki/SemEval
3 http://wordnet.princeton.edu/

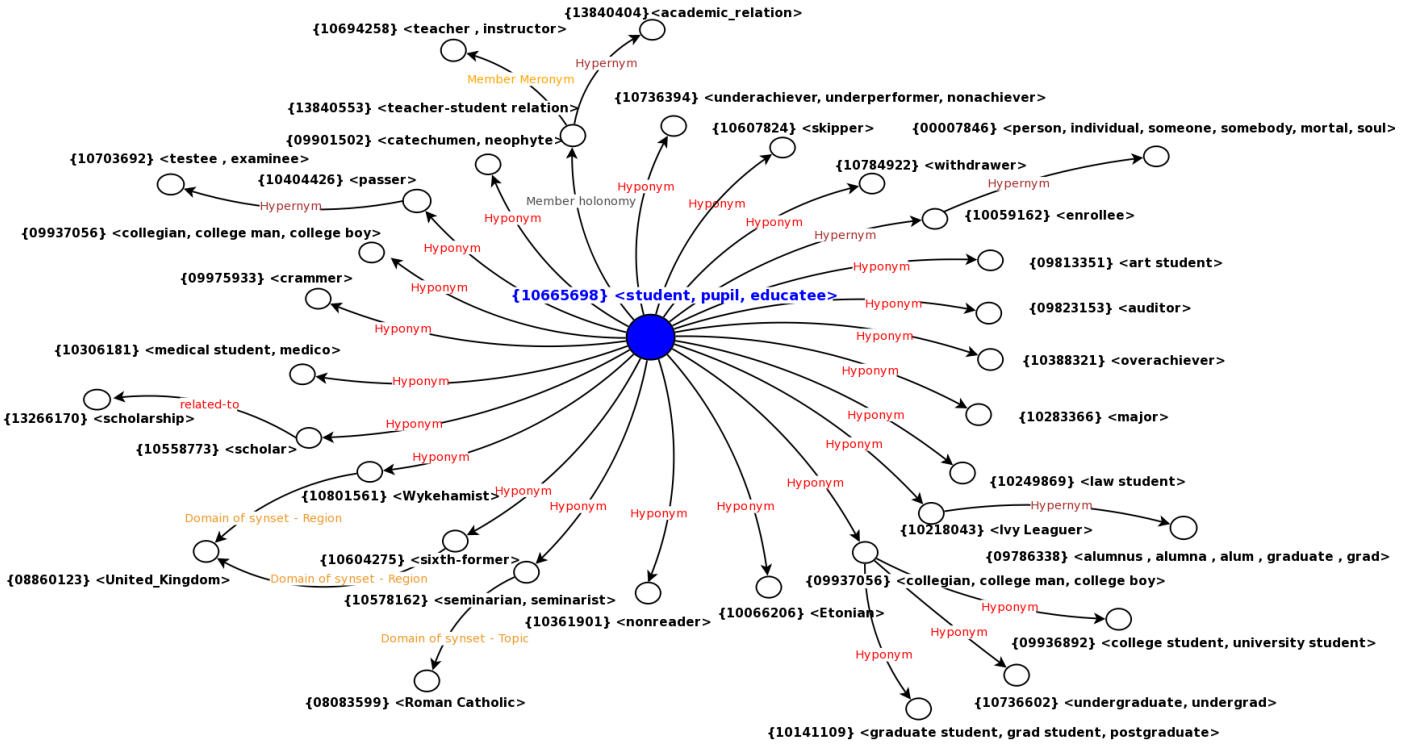

Figura 1 Muestra parcial de los synsets relacionados con el synset nominal <student, pupil, educatee>..

Su arquitectura está inspirada por teorías psicolingüisticas y computacionales sobre la memoria léxica humana. Contiene información codificada manualmente sobre nombres, verbos, adjetivos y adverbios del inglés, y esta organizada entorno a la noción de *synset*. Un synset es un conjunto de palabras de la misma categoría morfosintáctica que pueden ser intercambiadas en un contexto dado. Por ejemplo, <student, pupil, educatee> forman un synset porque pueden ser utilizados para referirse al mismo concepto. Además, un synset puede identificarse también por su offset[4]. Para <student, pupil, educatee> el offset es 10665698. Un synset es comúnmente descrito por una *gloss* o definición, que en el caso del synset anterior es *"a learner who is enrolled in an educational institution"*, y además, por un conjunto explícito de relaciones semánticas con otros synsets. Cada synset representa un concepto que está relacionado con otros conceptos mediante una gran variedad de relaciones semánticas, incluyendo hiperonimia, hiponimia, meronimia, holonimia, antonimia, etc. Los synsets están enlazados entre sí mediante relaciones léxicas y semántico-conceptuales. WordNet también codifica 26 tipos diferentes de relaciones semánticas. WordNet está disponible de modo público y gratuito para su descarga. La última versión accesible de WordNet es la 3.0. Su estructura lo convierte en una herramienta útil para la lingüística computacional y el procesamiento de lenguaje natural. Resulta evidente que WordNet se ha convertido en un estándar en el área de PLN. De hecho, WordNet es usado en todo el mundo como base para anclar distintos tipos de conocimiento semántico, incluyendo wordnets de otros idiomas (Vossen 98).

La versión de WordNet 3.0 tiene un total de 117.659 synsets y 235.402 relaciones semánticas[5].

Por ejemplo, la figura 1 muestra algunas de las relaciones que aparecen codificadas en WordNet del synset <student, pupil, educatee>. WordNet tiene codificado para este synset un total de 26 relaciones con otros synsets.

Adicionalmente, el grupo de Princeton distribuye un nuevo recurso con las definiciones de WordNet manualmente anotadas con el correspondiente sentido de WordNet[6]. Este recurso, que proviene del conjunto completo de glosas de WordNet 3.0, contiene un total de 652.017 sustantivos, verbos, adjetivos o adverbios. Sin embargo, de todas ellas, sólo 454.439 términos han sido desambiguadas manualmente (aproximadamente, un 70% del total). Esto quiere decir que aún quedan 197.578 ocurrencias contenidas en las glosas de WordNet sin su sentido correcto asignado (aproximadamente un 30% de las palabras).

Así, algunas palabras de la definición de <student, pupil, educatee> también han sido anotadas con su sentido correcto. En concreto, el sustantivo *learner* (que en wordNet es

4 Su posición en el fichero data.noun de la distribución de WordNet

5 Las relaciones simétricas sólo se contabilizan una vez.

6 http://wordnet.princeton.edu/glosstag.shtml

polisémica con dos sentidos) y el adjetivo *educational* (también con dos sentidos). Estas anotaciones nos permiten crear dos relaciones nuevas entre el synset definido <student, pupil, educatee> y los synsets de *learner* y *educational*. Nótese sin embargo que no todas las palabras de su definición han sido desambiguadas. En este caso, el verbo *enroll* y el sustantivo *institution*. Por último, el synset <student, pupil, educatee> también puede haber sido utilizado para anotar otras palabras de definiones de otros synsets. Por ejemplo, el synset <secondary_school, lyceum, lycee, Gymnasium, middle_school> definido como "*a school for students intermediate between elementary school and college; usually grades 9 to 12*". En realidad, el synset <student, pupil, educatee> aparece anotando 132 definiciones. De WordNet 3.0, Con ellas podemos crear 132 nuevas relaciones.

En resumen, para el synset <student, pupil, educatee> podemos crear como máximo[7] un total de 160 relaciones con otros synsets.

Así, si consideramos cada synset de WordNet como un nodo y cada relación entre synsets como un arco no dirigido, podemos crear un grafo con 117.536 vértices y 561.793 arcos.

B. Algoritmos SSI

SSI (Navigli & Velardi 2005) (del inglés Structural Semantic Interconnections) es un algoritmo de WSD basado en grafos. El algoritmo es muy simple y consiste en un paso de inicialización y un número de pasos iterativos.

Dada W, una lista ordenada de palabras que deben ser desambiguadas, el algoritmo SSI funciona de la siguiente manera: en el paso de inicialización, todas las palabras monosémicas son incluidas en una lista I que contiene las palabras que ya han sido interpretadas, mientras que las palabras polisémicas son incluidas en la lista P, donde estarán todas las palabras pendientes de desambiguar. En cada paso, la lista I se usa para desambiguar una palabra de la lista P, seleccionando el sentido de la palabra *más cercana* a las palabras ya desambiguadas de la lista I. Una vez seleccionado el sentido, esa palabra la eliminamos de la lista P y la incluimos en la lista I. El algoritmo termina cuando ya no quedan más palabras pendientes de desambiguar en la lista P.

Con el fin de medir la proximidad de un synset (de la palabra de P a ser desambiguada en cada paso) a un conjunto de synsets (aquellos sentidos ya interpretados en I), el algortimo SSI original utiliza una base conocimiento propietaria derivada semi-automáticamente que integra una ámplia variedad de recursos (Navigli, 2005). Esta base de conocimiento se utiliza para calcular las distancias entre los synsets del grafo. Sin embargo, con el fin de evitar la explosión exponencial de posibilidades, no se consideran todos los caminos posibles. El algoritmo usa una gramática libre de contexto sobre las relaciones entrenadas sobre un corpus anotado con sentidos (SemCor), para descartar caminos no apropiados y dar peso a los apropiados. De esta manera, el algoritmo no consideraba la totalidad de relaciones dentro del grafo.

SSI-Dijkstra (Cuadros & Rigau 2008) utiliza el algoritmo de Dijkstra para obtener el sentido más cercano a un conjunto dado de sentidos de un grafo. El algoritmo de Dijkstra es un algoritmo voraz que calcula la distancia del camino más corto entre un nodo de un resto de nodos de un grafo (Dijkstra 1959, Cormen et al. 2001). La biblioteca BoostGraph[8] se puede utilizar para calcular de manera muy eficiente la distancia más corta entre dos nodos dados en grafos muy grandes. Para construir el grafo, utilizaron distintas bases de conocimiento públicas alineadas a WordNet 1.6 (Fellbaum 1998) e integradas en el Multilingual Central Repository[9] (Atserias et al. 2004). El algortimo I muestra el pseudocódigo del algoritmo SSI-Dikjstra.

ALGORITMO I

ALGORITMO DE WSD SSI-DIJKSTRA

```
Algoritmo SSI-Dijkstra
  SSI (T : list of terms)
  for each {t ∈ T} do
     I[t] = ∅
     if t es monosemica then
        t := el unico sentido de t
     else
        P := P ∪ {t}
     end if
  end for
  repeat
     P' := P
     for each {t ∈ P} do
        MejorSentido := ∅
        MaximoValor := 0
        for each {sentido s de t} do
           W[s] := 0
           N[s] := 0
           for each {sentido s' ∈ I} do
              w := DijkstraShortestPath(s, s')
              if w > 0 then
                 W[s] := W[s] + (1/w)
                 N[s] := N[s] + 1
              end if
           end for
           if N[s] > 0 then
              NuevoValor := W[s]/N[s]
              if NuevoValor > MaximoValor then
                 MaximoValor := NuevoValor
                 MejorSentido := s
              end if
           end if
        end for
        if MaximoValor > 0 then
           I[t] := MejorSentido
           P := P/{t}
        end if
     end for
  until P ≠ P'
  return (I,P);
```

SSI-Dijkstra+ (Laparra et al., 2010) es una versión extendida de SSI-Dijkstra. En ausencia de monosémicos, SSI-Dijkstra no proporciona ningún resultado, ya que depende directamente del conjunto I de palabras interpretadas para dar una solución. SSI-Dijkstra+ es una versión mejorada del algoritmo SSI-Dijkstra en dos aspectos. En primer lugar, SSI-

7 Puede que haya relaciones repetidas, por ejemplo entre un hiperónimo y una relación de glosa.

8 http://www.boost.org/libs/graph

9 http://adimen.si.ehu.es/web/MCR

Dijkstra+ obtiene solución cuando los conjuntos de palabras por procesar no tienen monosémicos. Para ello, obtiene la palabra menos ambigua (la de menor grado de polisemia, o dicho de otro modo, la palabra con menos sentidos) y realiza una hipótesis acerca de su significado más correcto. La hipótesis se obtiene a partir del resultado de dos algoritmos diferentes que miden su distancia acumulada a P. En segundo lugar el algoritmo parece mejorar si para los verbos se utiliza además de I, los primeros sentidos de las palabras de P.

Nótese que todos los algoritmos SSI procesan una a una las palabras de P, y para cada una de ellas se debe comprobar cuál de sus sentidos está más próximo a los sentidos ya seleccionados en I. En el caso de SSI-Dijkstra, esto significa que se deberán aplicar tantas veces Dijkstra como sentidos distintos haya en las palabras de P.

C. UKB

UKB[10] (Agirre & Soroa 2009) es una biblioteca para trabajar en el campo de la desambiguación de palabras usando grafos. Esta biblioteca está codificada en C++ y también utiliza la librería BoostGraph. La versión actual de UKB 2.0 es muy eficiente incluso para grafos extremadamente grandes y se compone de clases para creación, lectura y procesamiento de grafos a partir de ficheros que contienen la base de conocimiento. UKB aplica el método PageRank personalizado sobre una base de conocimiento léxico para dar peso a los vértices de la base de conocimiento y así realizar la desambiguación (selecciona los sentidos de las palabras del contexto más pesados. Los detalles del método se describen en (Agirre & Soroa 2009). El algoritmo también se puede utilizar para calcular similitud léxica o relación entre palabras (Agirre et al. 2010).

Usando como base esta misma versión de UKB, nosotros hemos añadido al código nuevas clases y funciones para reimplementar e integrar de forma modular las funciones SSI-Dijkstra, SSI-Dijkstra+ y SSI-Dijkstra-Fast en UKB[11]. El código resultante está disponible bajo licencia GNU[12]

III. SSI-DIJKSTRA-FAST

Para optimizar el algoritmo SSI-Dijkstra, en lugar de calcular un Dijkstra para cada sentido de las palabras en P (y así obtener las distancias a los sentidos de I), proponemos cambiar la dirección del cálculo de distancias. Así, modificamos el algoritmo para obtener las distancias de los sentidos de I al resto de sentidos en P. Para ello, además, necesitaremos almacenar las distancias en una matriz auxiliar. En el algoritmo II se describe el pseudocódigo de la solución propuesta.

Como se puede apreciar la llamada del algoritmo dijkstra se elimina del proceso iterativo normal y se deja al inicio y al final de cada iteración. Con ello se puede mantener el ciclo de desambiguación pero realizando los cálculos de forma distinta. Primero se calculan las distancias desde los sentidos de I a los sentidos de las palabras en P. Es decir, por cada sentido de I se realiza una llamada a Dijkstra_shorttest_path. Las distancias de I a todos los sentidos de las palabras de P son almacenados en una matriz auxiliar. Luego, en cada iteración, para cada nuevo sentido seleccionado de P e introducido en I, se realiza una nueva llamada a Dijkstra_shortest_path. Las distancias del nuevo sentido introducido en I al resto de sentidos en P son también almacenamos en la matriz auxiliar de distancias. De esta manera, reducimos considerablemente la cantidad de llamadas a Dijkstra_shortest_path, ya que en total, ejecutamos esta función tantas veces como palabras haya en W (en lugar de una llamada a Dijkstra_shortest_path por cada sentido que encontremos en P).

ALGORITMO II
ALGORITMO DE WSD SSI-DIJKSTRA-FAST

Algorithm 1 SSI-Dijkstra-Fast

```
SSIDF (T: list of terms)
for each {t ∈ T} do
    I[t] = ∅
    if t is monosemous then
        I[t] := the only sense of t
    end if
    P := P ∪ {t}
end for
for each {i ∈ I} do
    for each {p ∈ P} do
        for each {sense j ∈ p} do
            d[i, j] = dijkstra_shortest_path(i, j)
        end for
    end for
end for
repeat
    P' := P
    for each {p ∈ P} do
        BestSense := ∅
        MaxValue := 0
        for each {sense j ∈ p} do
            W[s] := 0
            N[s] := 0
            for each {i ∈ I} do
                w := d[i, j]
                if w > 0 then
                    W[s] := W[s] + (1/w)
                    N[s] := N[s] + 1
                end if
            end for
            if N[s] > 0 then
                NewValue := W[s]/N[s]
                if NewValue > MaxValue then
                    MaxValue := NewValue
                    BestSense := s
                end if
            end if
        end for
        if MaxValue > 0 then
            I[t] := BestSense
            P := P \ t
            i := BestSense
            for each {p ∈ P} do
                for each {sense j ∈ p} do
                    d[i, j] = dijkstra_shortest_path(i, j)
                end for
            end for
        end if
    end for
until P ≠ P'
return (I,P);
```

10 http://ixa2.si.ehu.es/ukb

11 Las versiones originales estaban codificadas en Perl.

12 https://github.com/gblancog/ukb/archive/master.zip

IV. Marco de Evaluación

Para realizar la evaluación empírica del nuevo algoritmo SSI-Dijkstra-Fast también usaremos como conjunto de test una parte de las anotaciones manuales proporcionadas las glosas desambiguadas manualmente de WordNet 3.0. Para ello, hemos seleccionado aleatoriamente un subconjunto de 933 asignaciones manuales correspondientes a 933 glosas distintas del corpus *Semantically Tagged glosses*[13] para formar un conjunto de test. Así, hemos seleccionado para cada una de las 933 glosas seleccionadas del test una sola de sus palabras desambiguadas. El test consiste en determinar el sentido correcto de la palabra escogida.

Para ilustrar el proceso de creación del corpus vamos a emplear el concepto <automation$_n^2$> que mostramos a continuación:

offset: 14574504
synset: <automation$_n^2$>
glosa: *the condition of being automatically operated or controlled*
ejemplo: *automation increases productivity*
hiperónimo: <condition$_n^1$ status$_n^2$>
Relacionada: <automatize$_v^2$ automatise$_v^2$ automate$_v^1$>
glosa: <operate$_v^3$ control$_v^3$>
glosa: <mechanically$_r^1$ automatically$_r^2$>
glosa^{-1}: <perestroika$_n^1$>

Este concepto tiene dos relaciones codificadas, de las cuales una es un hiperónimo y otra es una derivación morfológica. Además, las relaciones desambiguadas automáticamente de las glosas desambiguadas de WordNet 3.0 contemplan dos relaciones (glosa), y a su vez, <automationn^2> aparece en una sola glosa (glosa^{-1}) (<perestroika$_n^1$>). Obsérvese que sólo dos palabras de su glosa han sido anotadas con su sentido (condition$_n$ y control$_v$). Así, este sentido está relacionado con un total de cinco conceptos distintos.

En particular, la palabra *operate*$_v$ de esta glosa fue seleccionada al azar para formar parte del conjunto de evaluación. En el corpus *Semantically Tagged glosses* de WordNet 3.0, el sentido asignado para esta palabra en esta glosa es operate$_v^3$ con el offset 01224744, la glosa "*handle and cause to function"* y los ejemplos "*do not operate machinery after imbibing alcohol; control the lever;"*.

De las 933 glosas seleccionadas aleatoriamente, 768 son glosas de sentidos nominales, 91 adjetivales, 64 verbales y 10 adverbiales. En total, las glosas de este test forman un corpus con 5.344 palabras de categoría abierta (sustantivos, verbos, adjetivos o adverbios). De este modo, este corpus lo conforman 3.285 sustantivos, 1.110 adjetivos, 735 verbos y 214 adverbios[14]. De media, este corpus tiene un total de 8.79 términos por glosa. Además, partimos de la segmentación, tokenización y desambiguación morfosintáctica que proporciona la propia versión del corpus *Semantically Tagged glosses* de WordNet 3.0, en el que por cierto, aparecen catorce errores de asignación de etiquetas morfosintácticas.

A continuación presentamos el contexto del synset <automation$_n^2$> en el formato que requiere la biblioteca UKB:

```
ctx_14574504-n(operate#v#wf6#01224744-v)
14574504-n#n#wf0#2 condition#n#wf2#1 be#v#wf4#1
    automatically#r#wf5#1 operate#v#wf6#1
    control#v#wf8#1
```

Este contexto está compuesto de dos líneas. La primera es el identificador del contexto, que corresponde a la glosa del synset nominal 14574504. Además, el identificador del contexto también codifica el término que vamos a testear de esta glosa. Cada término viene separado por el carácter "#". Así, se indica que el término a desamiguar es el verbo "operate". Este verbo se corresponde con el sexto término de la glosa y el sentido correcto del verbo "operate" en este contexto es el que se coresponde con el offset 01224744. La segunda línea se corresponde con los términos a desambiguar de la definición. Siempre proporcionamos el sentido de la glosa a desambiguar (en este caso 14574504). El cuarto campo de cada término es un código de control. Si es 2, UKB interpreta que el sentido no debe desambiguarse. Si es 1, UKB debe desambiguarlo.

Para evaluar los resultados hemos usado el software de evaluación estándard de las competiciones SensEval scorer2[15]. En estas competiciones los resultados se miden en términos de precisión (número de respuestas acertadas con respecto a las contestadas), recall (número de respuestas acertadas sobre el total) y F1 (media harmónica entre precisión y recall).

Finalmente, para simular un entorno lo más justo posible, tambien hemos creado el grafo con el conocimiento de WordNet y sus glosas desambiguadas, pero sin incluir ninguna de las ya desambiguadas en las 933 glosas del test. En total son 5.283 arcos que no han sido incluidos en el grafo. Por tanto, el conocimiento que proporcionan estas definiciones no está incluido en el grafo resultante. Con ello, el grafo resultante tiene 117.536 vértices y 557.156 arcos no dirigidos. Este escenario pretende simular el caso de integrar nuevas definiciones (sin desambiguar) en la estructura de WordNet 3.0.

V. Pruebas Empíricas

Hemos realizado dos tipos de pruebas. Las primeras para verificar que los resultados de SSI-Dijkstra-Fast tanto de precisión (precision) como de cobertura (recall) fueran los mismos o muy similares[16] a los del SSI-Dijkstra original. Las segundas para verificar su mejora en tiempos de ejecución. Todos las pruebas han sido realizadas en una misma máquina Scientific Linux con arquitectura x86_64 y CPUs Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz. Para cada sistema hemos

13 http://wordnet.princeton.edu/glosstag.shtml

14 El corpus está disponible en: http://adimen.si.ehu.es/~rigau/wnet30+g.v3.random.1.contexts

15 http://www.senseval.org/senseval3/scoring

16 Pueden haber discrepancias cuando hay empates en distintos valores del algoritmo.

medido el tiempo de usuario que devuelve el comando time de Linux (tiempo dedicado por la CPU al procesamiento del programa).

La tabla I muestra los resultados de precisión, recall , F1 y tiempo empleado por cada sistema.

TABLA I
RESULTADOS DE LOS DISTINTOS ALGORITMOS.

Sistema	**Precisión**	**Recall**	**F1**	**Tiempo**
SSID	0.840	0.827	0.834	14m06.422s
SSID+	0.830	0.818	0.824	14m09.000s
SSID-Fast	0.830	0.818	0.824	5m56.986s
UKB w2w	0.873	0.860	0.866	11m36.747s

Como esperábamos, los resultados de precisión, recall y F1 son muy parecidos para SSI-Dijkstra, SSI-Dijkstra+ y SSI-Dijkstra-Fast. Sin embargo, la versión optimizada de SSI-Dijkstra realiza la misma tarea en menos de la mitad de tiempo. En realidad, SSI-Dijkstra-Fast ha procesado un contexto cada 0.38 segundos (en lugar de los 0.91 segundos de SSI-Dijkstra original).

Incluimos también en la comparativa los resultados de UKB2.0 con la opción de mayor precisión (ppr_w2w). Como vemos, aunque los resultados de UKB son algo mejores, su tiempo de ejecución es casi el doble que el de SSI-Dijkstra-Fast (0.75 segundos por contexto).

V. CONCLUSIONES

En este trabajo hemos presentado SSI-Dijkstra-Fast, una versión optimizada del algoritmo SSI-Dijkstra para la resolución de la ambigüedad semántica de la palabras en un contexto (en inglés Word Sense Disambiguation).

Para facilitar la comparación entre distintos algoritmos de WSD basados en grafos, hemos reimplementado los algoritmos SSI-Dijkstra, SSI-Dijkstra+ y SSI-Dijkstra-Fast integrándolos en la biblioteca UKB 2.0. El código resultante está disponible bajo licencia GNU[17]

En pruebas sobre un mismo corpus y máquina hemos demostrado empíricamente que a) los resultados de SSI-Dijkstra-Fast son muy similares a los del SSI-Dijkstra original y b) que en ese mismo corpus SSI-Dijkstra-Fast realiza la misma tarea de desambiguación en menos de la mitad de tiempo. También hemos comparado los resultados de la versión optimizada de SSI-Dijkstra con el algoritmo de UKB que da mejores resultados. Aunque los resultados de UKB son algo mejores, su tiempo de ejecución es casi el doble que el de SSI-Dijkstra-Fast.

Cabe destacar que éste nivel de mejora es variable dependiendo del grado de polisemia de los contextos a desambiguar. A mayor polisemia, mayor porcentaje de mejora.

En un próximo futuro, nos planteamos aplicar estos mismos algoritmos a otras bases de conocimiento, por ejemplo Wikipedia o BabelNet (Navigli & Ponzeto 2012), además de en otros idiomas como el castellano, también integrado en el Multilingual Central Repository (Gonzalez-Agirre et al. 2012).

17 https://github.com/gblancog/ukb/archive/master.zip

AGRADECIMIENTOS

Queremos agrader a Aitor Soroa la ayuda y consejos prestados sobre la librería UKB. Este trabajo ha sido posible gracias al apoyo del proyecto europeo NewsReader (ICT-2011-316404) y al proyecto SKaTer (TIN2012-38584-C06).

BIBLIOGRAFÍA

Agirre E., & Edmonds P. G. (Eds.). (2006). *Word sense disambiguation: Algorithms and applications* (Vol. 33). Springer Science+ Business Media.

Agirre E., Baldwin T., & Martinez D. (2008). Improving Parsing and PP Attachment Performance with Sense Information. In ACL (pp. 317-325).

Agirre E., & Soroa A. (2009). Personalizing pagerank for word sense disambiguation. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (pp. 33-41). Association for Computational Linguistics.

Agirre E, Cuadros M., Rigau G. & Soroa A. (2010) Exploring Knowledge Bases for Similarity. Proceedings of LREC 2010. Valletta, Malta.

Atserias J., Villarejo L., Rigau G., Agirre E., Carroll J., Magnini B., Vossen P. (2004) The MEANING Multilingual Central Repository. In Proceedings of the Second International Global WordNet Conference (GWC'04). Brno, Czech Republic.

Carpuat M., & Wu D. (2007). Improving Statistical Machine Translation Using Word Sense Disambiguation. In EMNLP-CoNLL (pp. 61-72).

Chai, J. Y., & Biermann, A. W. (1999). The Use of Word Sense Disambiguation in an Information Extraction System. In AAAI/IAAI (pp. 850-855).

Cormen T. H., Stein C., Leiserson C. E., & Rivest R. L. (2001). Introduction to Algorithms. The MIT press.

Cuadros M., & Rigau G. (2008). Knownet: Building a large net of knowledge from the web. In Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1 (pp. 161-168). Association for Computational Linguistics.

Dale R., Moisl H., & Somers H. (2000) *Handbook of natural language processing*. CRC Press

Dijkstra E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik* **1**: 269–271.

Edmonds, P., & Kilgarriff, A. (2002). Introduction to the special issue on evaluating word sense disambiguation systems. *Natural Language Engineering*, *8*(4), 279-291.

Fellbaum, C. (Ed.). (1998). *WordNet: an electrical lexical database*. The MIT Press.

Gonzalez-Agirre A., Laparra E. and Rigau G. Multilingual Central Repository version 3.0. (2012) 8th international conference on Language Resources and Evaluation (LREC'12). Istambul, Turkey.

Ide, N., & Véronis, J. (1998). Introduction to the special issue on word sense disambiguation: the state of the art. *Computational linguistics*, *24*(1), 2-40.

Jurafsky, D., Martin, J. H., Kehler, A., Vander Linden, K., & Ward, N. (2000). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition* (Vol. 2). Upper Saddle River: Prentice Hall.

Laparra E., Rigau G., & Cuadros M. (2010). Exploring the integration of WordNet and FrameNet. In Proceedings of the 5th Global WordNet Conference (GWC 2010), Mumbai, India.

Manning C. D., & Schütze H. (1999). *Foundations of statistical natural language processing* (Vol. 999). Cambridge: MIT press.

Mihalcea R. (2005). Unsupervised large-vocabulary word sense disambiguation with graph-based algorithms for sequence data labeling. In Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing (pp. 411-418). Association for Computational Linguistics.

Navigli R. (2005). Semi-Automatic Extension of Large-Scale Linguistic Knowledge Bases. In FLAIRS Conference (pp. 548-553).

Navigli R., & Velardi P. (2005). Structural semantic interconnections: a knowledge-based approach to word sense disambiguation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(7), 1075-1086.

Navigli R., & Lapata M. (2007). Graph Connectivity Measures for Unsupervised Word Sense Disambiguation. In IJCAI (pp. 1683-1688).

Navigli R. (2009). Word sense disambiguation: A survey. ACM Computing Surveys (CSUR), 41(2), 10.

Navigli R. & Ponzetto S. BabelNet: The Automatic Construction, Evaluation and Application of a Wide-Coverage Multilingual Semantic Network. *Artificial Intelligence*, 193, Elsevier, 2012, pp. 217-250.

Ravin, Y. & Leacock, C. (Eds.). (2000). *Polysemy: Theoretical and computational approaches*. Oxford: Oxford University Press.

Sinha R., & Mihalcea R. (2007). Unsupervised graph-basedword sense disambiguation using measures of word semantic similarity. In Semantic Computing, 2007. ICSC 2007. International Conference on (pp. 363-369). IEEE.

Stevenson M., & Wilks Y. (2003). *Word sense disambiguation*. The Oxford Handbook of Comp. Linguistics, 249-265.

Vossen P. (Ed.). (1998). EuroWordNet: a multilingual database with lexical semantic networks. Boston: Kluwer Academic.

Zhong Z., & Ng H. T. (2012). Word sense disambiguation improves information retrieval. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1 (pp. 273-282). Association for Computational Linguistics.