TITULACIÓN: Ingeniería Técnica en Informática de Sistemas

SSI-Dijkstra+: Desarrollo de un sistema de resolución de la ambigüedad semántica basada en grafos.

Alumno/a: D./Dña.Aritza Ledesma Ruiz
Director/a: D./Dña.German Rigau Claramunt

Proyecto Fin de Carrera, julio de 2012

Índice de contenido

1- Introducción

1.1-Presentación

Este documento es la memoria de un Proyecto de Final de Carrera de la Ingeniería Técnica en Informática de Sistemas en la Facultad de Informática de San Sebastián que se encuadra en el procesamiento del lenguaje natural (PLN), el campo de la semántica y la desambiguación de palabras. Está realizado por Aritza Ledesma y dirigido por Germán Rigau Claramunt (profesor asociado en la Facultad de Informática de San Sebastián, Universidad Pública Vasca (UPV/EHU)).

1.2-Motivación

Este proyecto surge de la necesidad de actualizar y corregir un software existente, llamado SSI-Dijstra+, realizado por Gorka Blanco en un PFC anterior. En esta memoria daremos una idea del funcionamiento en general y los pasos que se han seguido para elaborar una nueva versión del mismo. Para tener una noción básica del programa, con el que vamos a trabajar, expondremos un breve ejemplo de su funcionamiento:

Dado un contexto compuesto por un conjunto de palabras, de las cuales desconocemos su significado, nuestro algoritmo intentará asignar a cada palabra el significado más apropiado, en función de su contexto. Ha este proceso se le llama desambiguación. Además, para cada palabra disponemos de su categoría morfosintáctica: nombres, verbos, adjetivos o adverbios. Es necesario ya que hay palabras que pueden tener varias categorías. Ésto sucede con market, ya que puede ser nombre (mercado) o verbo (vender). Una vez desambiguadas las palabras, obtendremos un identificador, que a su vez está asociado a una única definición de un diccionario. Para ilustrar un poco más esta explicación, presentamos un pequeño ejemplo de una solución obtenida al desambiguar una serie de palabras seleccionadas:

- *Market (verbo)* →02298160 →participar en la promoción comercial, venta o distribución
- *Fruit (nombre)* →13134947 →el cuerpo reproductivo maduro de una planta de semillas
- *Meat (nombre)* → 07649854 → la carne de los animales (incluidos los peces, las aves y los caracoles) utilizada como alimento
- *Supermarket* (nombre) → 04358707 → un gran auto-servicio de comestibles, productos lácteos y artículos para el hogar

Para realizar dicha desambiguación, hemos usado un método de desambiguación, que usa una base de conocimiento en forma de grafo o red semántica llamado SSI-Dijkstra.

2- DOP: Documento de Objetivos del Proyecto

2.1.- Objetivos

El objetivo del proyecto es corregir, actualizar e implementar nuevas opciones al algoritmo SSI-Dijkstra . Como hemos comentado anteriormente, el algoritmo no funcionaba de forma correcta, por lo que se ha tenido que corregir.

SSI-Dijkstra+ está basado en una versión antigua del UKB (más concretamente en la 1.5), por lo que se ha decido actualizarlo, para poder tenerlo en la versión final. Como es habitual, la nueva versión del UKB dispone de nuevas funcionalidades, así que se tuvo que modificar el SSI-Dijkstra para que las contemplara.

2.2-Método de trabajo

Para desarrollar el proyecto, se ha elegido el Proceso Unificado de Desarrollo de Software (PUD), siendo uno de los métodos recomendados para el desarrollo del sistema de software.

El PUD se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y ser iterativo e incremental. Ahora pasaremos a explicar con más claridad, las características principales del PUD.

- **Dirigido por los casos de uso**

En el Proceso Unificado de Desarrollo, los casos de uso se utilizan para capturar los requisitos funcionales de la aplicación. Además, en cada iteración se toma un conjunto de casos de uso o escenarios, para desarrollar cada una de las distintas disciplinas: diseño, implementación, prueba, etc

- **Centrado en la arquitectura**

El Proceso Unificado de Desarrollo asume que, no existe un componente único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples componentes que definen la arquitectura de software de un sistema. La analogía con la construcción es clara, cuando construyes un edificio existen diversos componentes que facilitan los distintos servicios del mismo: electricidad, fontanería, etc.

- **Iterativo e Incremental**

El Proceso Unificado de Desarrollo contempla un marco de desarrollo iterativo e incremental, compuesto de cuatro fases denominadas: inicio, elaboración, construcción y transición. Cada una de éstas fases está, a su vez, dividida en una serie de iteraciones. Estas iteraciones añaden o mejoran las funcionalidades del sistema en desarrollo.

Cada una de estas iteraciones se divide, a su vez, en una serie de disciplinas que recuerdan a las definidas en el ciclo de vida clásico o en cascada: análisis de requisitos, diseño, implementación y prueba. Aunque todas las iteraciones suelen incluir algún desarrollo en casi todas las disciplinas, el grado de esfuerzo dentro de cada una de ellas varía a lo largo del proyecto.

- **Enfocado en los riesgos**

El Proceso Unificado de Desarollo requiere que el equipo del proyecto se centre en identificar los riesgos críticos en una etapa temprana del ciclo de vida. Los resultados de cada iteración, en especial los de la fase de Elaboración, deben ser seleccionados en un orden que asegure que los riesgos principales son considerados en primer lugar.

Cada vez que se finaliza una fase, se concreta un día para realizar una reunión para comentar la fase terminada y concretados nuevos objetivos de la próxima iteración. Éstas reuniones se realizaban tras finalizar los objetivos, ya que no se iban a introducir nuevos si no se había podido finalizar los anteriores. En caso de duda, el alumno disponía de medios suficientes para intentar solventarla mediante: correo, internet, documentación... Si por esos medios no se podía llegar a una solución, se

acordaba una reunión con el director para poder solucionarlo cuanto antes y seguir avanzando. También se muestra al director, el progreso del proyecto y el funcionamiento del mismo. Por ese motivo, se muestra la ejecución del mismo.

2.3-Alcance

2.3.1-Recursos humanos

Los recursos humanos, para este proyecto, han sido el director y el alumno. El profesor iba marcando los objetos a cumplir, siendo responsabilidad del alumno completarlos en el tiempo establecido para ello.

2.3.2-Recursos materiales

El alumno disponía de un ordenador portátil con Windows 7 y Ubuntu instalados, así como material de oficina (impresora, escáner....). También se disponía de varias memorias USB en las cuales hacer copias de seguridad, por si el ordenador portátil fallaba. Si el ordenador portátil fallaba, también se contaba con un ordenador de sobremesa para poder seguir con el proyecto.

El profesor contaba con su propio ordenador y su material de oficina para poder explicar al alumno los diferentes objetivos.

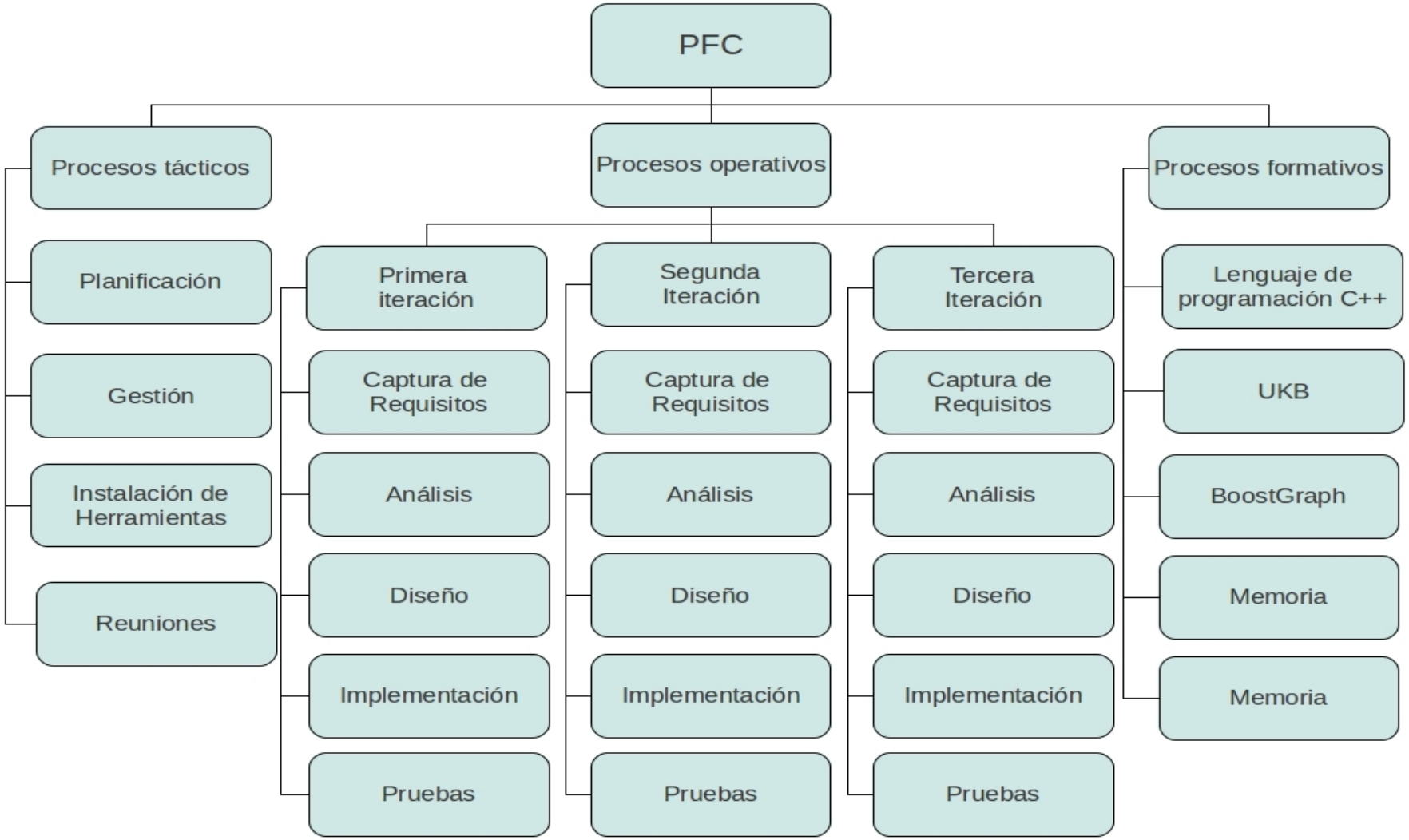
El lugar elegido para el desarrollo ha sido la casa del alumno en la que se dispone de conexión Wi-Fi y de material de oficina necesario. El lugar elegido para las reuniones ha sido el despacho del director.

2.3.3-Recursos económicos

El coste de este proyecto es cero, ya que al ser un proyecto de fin de carrera, no había recursos económicos. Todo el material utilizado en él ya estaba disponible antes de empezar ha realizarlo (ordenador portátil, memorias usb, impresora....), por lo que no se tuvo que hacer ninguna inversión. El único material nuevo fue la parte software, pero se decidió hacerlo con software libre. Por tanto no se tuvo que pagar ninguna licencia.

2.3.4-Diagrama EDT

El EDT es una descomposición jerárquica del trabajo a ser ejecutado. Cada nivel del EDT detalla con más precisión las distintas tareas del proyecto. El EDT es una herramienta fundamental en la gestión de proyectos.


PFC
Procesos tácticos
Procesos operativos
Procesos formativos
Planificación
Gestión
Instalación de Herramientas
Reuniones
Primera iteración
Captura de Requisitos
Análisis
Diseño
Implementación
Pruebas
Segunda Iteración
Captura de Requisitos
Análisis
Diseño
Implementación
Pruebas
Tercera Iteración
Captura de Requisitos
Análisis
Diseño
Implementación
Pruebas
Lenguaje de programación C++
UKB
BoostGraph
Memoria
Memoria

2.3.5-Procesos

Dividimos los procesos en las siguientes categorías: tácticos, operativos y formativos. A continuación se describirá las diferentes partes que conforman dichos procesos.

Procesos formativos

- Lenguaje programación C++:

Descripción: Aprendizaje del lenguaje C++ para la realización del proyecto, ya que el alumno lo desconoce.
Tiempo: 15 horas

- Librería BoostGraph:

Descripción: Aprendizaje de las funciones básicas de dicha librería.
Tiempo: 15 horas

- UKB:

Descripción: Aprendizaje exhaustivo del funcionamiento y funciones del UKB. Ésto incluye la documentación y el código.
Tiempo: 20 horas

- Memoria

Descripción: Realización de la memoria basada en el proyecto fin de carrera.
Tiempo: 50 horas

- Presentación:

Descripción: Realización de la presentación del proyecto fin de carrera.
Tiempo:7 horas

Procesos tácticos

- Planificación

Descripción: Tomar las decisiones pertinentes para la planificación del proyecto, incluyendo la memoria.
Tiempo: 10 horas

- Gestión

Descripción: llevar a cabo las acciones para gestionar el proyecto.
Tiempo: 5 horas

- Instalación de herramientas:

Descripción: Instalar todos los programas necesarios para realizar el proyecto.
Tiempo: 10 horas

- Reuniones:

Descripción: Tiempo usado para realizar las reuniones con el director para decidir nuevos objetivos, ver el estado actual del proyecto...
Tiempo: 20 horas

Procesos operativos

- Primera iteración

Descripción: Esta iteración está comprendida desde el inicio del proyecto hasta la corrección de los algoritmos SSI-Dijkstra y el SSI-Dijkstra+.

a) Captura de requisitos

Descripción: Captura de requisitos de la primera iteración.
Tiempo estimado: 6 horas

b) Análisis

Descripción: Análisis de la primera iteración.
Tiempo estimado: 5 horas

c) Diseño

Descripción: Diseño de la primera iteración.
Tiempo estimado: 7 horas

d) Implementación

Descripción: Implementación de la primera iteración.
Tiempo estimado: 10 horas

e) Pruebas

Descripción: Pruebas de la primera iteración
Tiempo estimado: 6 horas

- Segunda iteración

Descripción: En esta iteración se actualiza el proyecto pasando de la versión del UKB ukb-0.1.5r1, a la versión ukb-0.1.6.

a) Captura de Requisitos

Descripción: Captura de requisitos de la segunda iteración.
Tiempo estimado: 5 horas

b) Análisis

Descripción: Análisis de la segunda iteración.
Tiempo estimado: 5 horas

c) Diseño

Descripción: Diseño de la segunda iteración.
Tiempo estimado: 7 horas

d) Implementación

Descripción: Implementación de la segunda iteración.
Tiempo estimado: 10 horas

e) Pruebas

Descripción: Pruebas de la segunda iteración
Tiempo estimado: 5 horas

- Tercera iteración

Descripción: En esta iteración se les añaden nuevos tipos de palabras y opciones a los algoritmos SSI-Dijkstra y el SSI-Dijkstra+. También se actualiza a la última versión disponible del UKB.

a) Captura de requisitos

Descripción: Captura de requisitos de la tercera iteración.

Tiempo estimado: 5 horas

b) Análisis

Descripción: Análisis de la tercera iteración.

Tiempo estimado: 4 horas

c) Diseño

Descripción: Diseño de la tercera iteración.

Tiempo estimado: 5 horas

d) Implementación

Descripción: Implementación de la tercera iteración.

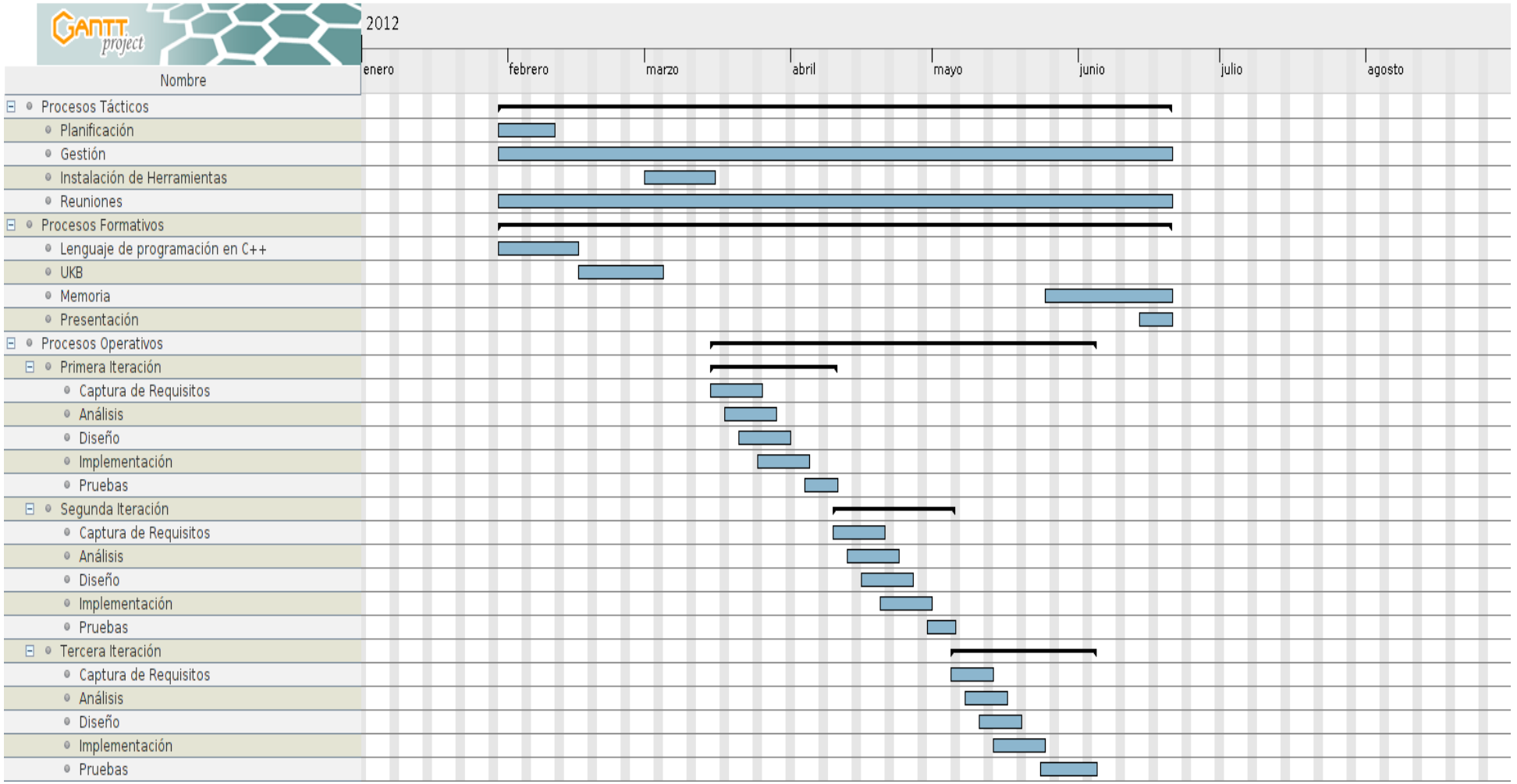
Tiempo estimado: 10 horas

e) Pruebas

Descripción: Pruebas de la tercera iteración

Tiempo estimado: 7 horas

2.4- Planificación temporal: Diagrama de gantt



2.5-Plan de contingencia

A continuación, se exponen los diferentes riesgos que pueden darse a lo largo del proyecto. Para poder actuar de una manera rápida y eficaz, es importante tenerlos identificados, ya que puede significar el éxito o el fracaso del proyecto. Ahora nos dispondremos a explicar una serie de riesgos junto a sus soluciones:

- *Pérdida de documentación*

Descripción: Pérdida de la documentación, ya sea física o por no poder tener acceso a ella
Probabilidad: Baja
Gravedad: Media
Solución: La documentación física se podía conseguir imprimiendo otra vez lo mismo. Si son explicaciones sobre papel hechas por el director, se le comunicaría para que pueda repetirlas. El resto de la documentación proporcionada por el director se obtiene del correo del alumno, ya que se mandaba por dicho medio. Toda la documentación obtenida por el alumno se guardaba en una copia de seguridad.

Riesgos Tácticos

- *Imposibilidad para reunirse*

Descripción: La imposibilidad por parte del alumno o del directo para reunirse.
Probabilidad: Baja
Gravedad: Media
Solución: Utilizar el correo o la videoconferencia (skype), con el cual, el alumno y el director puedan seguir comunicándose para avanzar con el proyecto.

- *Retraso en la tarea*

Descripción: Retraso en alcanzar los objetivos marcados por el director.
Probabilidad: Media
Gravedad: Media
Solución: Resolver los problemas que han originado el retraso y replanificar.

Riesgos Operativos

- *Pérdida del proyecto*

Descripción: Pérdida de todos los datos relacionados con el proyecto
Probabilidad: Baja
Gravedad: Alta
Solución: Usar la copia de seguridad más reciente guardado en una memoria USB.

- *Ordenador estropeado*

Descripción: El ordenador actual de trabajo deja de funcionar
Probabilidad: Media
Gravedad: Media
Solución: Se carga la copia de seguridad más reciente en un segundo ordenador, que estaba disponible desde el principio del proyecto.

- *Problemas a la hora de instalar el software*

Descripción: Problemas que puedan surgir a la hora de instalar el software necesario para la elaboración del proyecto.
Probabilidad: Media
Gravedad: Media
Solución: Reunirse con el director para intentar solucionarlo lo antes posible si no ha sido posible solucionarlos buscando información por internet, manuales.....

- *Copia de seguridad estropeada*

Descripción: La unidad USB en la que se guardaba la copia de seguridad deja de funcionar.
Probabilidad: Baja
Gravedad: Alta
Solución: Tener siempre dos memorias USB en las que guardar una copia de seguridad, ya que al no ser un proyecto de mucha envergadura, es posible realizarlas en un pequeño lapso de tiempo. También sustituir la memoria USB estropeada por otra, y realizar la copia de seguridad.

Riesgos Formativos

- *Dificultad a la hora de manejar el software*

Descripción: Problemas surgidos a la hora de manejar el software del proyecto por desconocimiento del alumno.
Probabilidad: Media
Gravedad: Media
Solución: Buscar información en internet, manuales... para intentar solucionarlos cuanto antes. Si el director tiene conocimiento sobre el mismo, preguntarle directamente.

2.6-Análisis de factibilidad

El tiempo total planificado es de 249 horas, que sobrepasa las horas propuestas para un Proyecto Final de Carrera de la Ingeniería Técnica en Informática de Sistemas, siendo éstas 150 horas. Aun así, ya se intuía que serían necesarias más horas para la realización del mismo.

Tras analizar las diferentes tareas, el tiempo estimado para su realización y los diferentes riesgos que puede haber, teniendo en cuenta las soluciones propuestas, vemos factible realizar el proyecto en el tiempo establecido.

3- Antecedentes

3.1- Redes semánticas

Una red semántica es una forma de representación de conocimiento lingüístico, en la que los conceptos y sus interrelaciones se representan mediante un grafo. Los conceptos, ó unidades léxicas, se representan mediante los nodos del grafo, y las interrelaciones que existen entre conceptos están representadas por las aristas dirigidas del grafo. En el caso de que no existan ciclos entre conceptos, los grafos también pueden ser dibujados en forma de árbol.

En las redes semánticas, un concepto importante es la distancia semántica que se expresa en relación al número y tipo de enlaces que separan un nodo de otro.

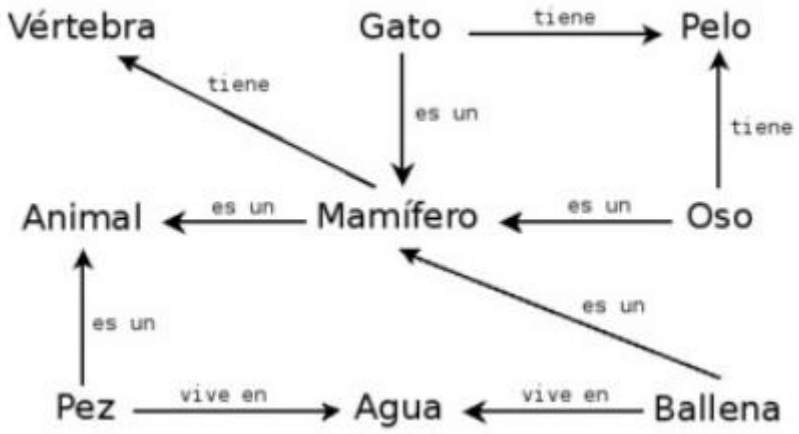


Figura 1: Ejemplo de red semántica

En la Imagen 2 podemos ver representado cierto conocimiento acerca de los mamíferos. En ella identificamos los elementos característicos de las redes semánticas como los nodos (mamífero, animal, oso, pez,...), las relaciones (es un, tiene, vive en) así como las distancias entre ellos.

Existen diversos tipos de relaciones semánticas. Por ejemplo, la sinonimia/antonimia (relación entre sinónimos y antónimos), hiponimia/hiperonimia (relación entre subordinados y superordinados), la meronimia/holonimia (relación entre partes y conjuntos), entre otras.

Una de las redes semánticas más conocidas es WordNet.[1]

3.1.1- Wordnet

Desarrollada en la universidad de Princeton, WordNet2 (Miller et al., 1990) es una base de datos léxica de dominio general para el inglés, que actualmente constituye uno de los recursos léxicos más utilizados en el área del Procesamiento del Lenguaje Natural (PLN). WordNet (WN) ha sido creado para intentar organizar la información léxica por significados. A diferencia de los diccionarios convencionales, donde ésta información está organizada por la forma de las unidades léxicas.

WN está estructurada como una red semántica cuyos nodos, denominados synsets (synonym sets, o conjuntos de sinónimos) constituyen su unidad básica de significado. Cada uno de ellos se compone de un conjunto de las lexicalizaciones que representan un sentido y se identifica mediante un "offset" (byte) y su correspondiente PoS (Part of Speech); que puede ser (n) para nombres, (v) para verbos, (a) para adjetivos y (r) para adverbios. Por ejemplo:

02152053#n fish#1
01926311#v run#1
02545023#a funny#4
00005567#r automatically#1

El número que aparece después de cada palabra indica el número de sentido que representa el synset. Una palabra puede ser polisémica, esto es, tener varios significados. Por ejemplo:

- 02152053#n fish#1; representa la palabra fish con el sentido de pez como animal vertebrado acuático.

- 07775375#n fish#2; representa la palabra fish con el sentido de pescado como alimento.

También puede ser que varias palabras tengan el mismo significado, esto es, que sean

1 http://wordnet.princeton.edu

sinónimas. En Wordnet están representados en el mismo synset:

- 02383458#n car#1, auto#1, automobile#1, machine#4, motorcar#1; representa el vehículo de cuatro ruedas.

Todos los synsets incluyen una glosa a modo de definición similar a la del diccionario tradicional, que describe el significado del concepto de forma explícita. Por ejemplo:

- 02383458#n car#1; 4-wheeled motor vehicle; usually propelled by an internal combustion engine: he needs a car to get to work;

Como ya se expuso anteriormente, WordNet está estructurada como una red semántica. Además de la información que pueden proporcionar los sinónimos (componentes del synset) y la glosa, tenemos que tener en cuenta los arcos de la red semántica. Éstos establecen diferentes relaciones entre los synsets, por ejemplo:

Hiperonimia: Es el término genérico usado para designar a una clase de instancias específicas. Y es un hyperónimo de X, si X es una clase de Y.

- tree#n#1 HYPERONYM oak#n#2

Hiponimia: Es el término específico usado para designar el miembro de una clase, X es un hipónimo de Y, si X es una clase de Y. En el caso de los verbos se denomina Troponimia.

- oak#n#2 HYPONYM tree#n#1

Antonimia: Es la relación que enlaza dos sentidos con significados opuestos.

- active#a#1 ANTONYM_OF inactive#a#2

- inactive#a#2 ANTONYM_OF active#a#2

Meronimia: Es la relación que se define como componente de, substancia de, o miembro de algo, X es merónimo de Y si X es parte de Y.

- car#n#1 HAS_PART window#n#2
- milk#n#1 HAS_SUBSTANCE protein#n#1
- family#n#1 HAS_MEMBER child#n#2

Holonimia: Es la relación contraria a la meronimia, Y es holónimo de X si X es una parte de Y. window#n#2 PART_OF car#n#1

- protein#n#1 SUBSTANCE_OF milk#n#1
- child#n#2 MEMBER_OF family#n#2

El conocimiento de Wordnet va aumentando en cada una de sus versiones según se van incluyendo nuevos synsets. Vamos a mostrar unas tablas que reflejan en qué estado se encuentra el WordNet en su versión 3.0.

Pos	**Únicos Strings**	**Synsets**	**Pares de sentido-palabra**
Nombre	117798	82115	146312
Verbo	11529	13767	25047
Adjetivo	21479	18156	30002
Adverbio	4481	3621	5580
Total	155287	117659	206941

Tabla 1: Número de palabras, synsets y sentidos

Pos	Palabras y sentido monosémicos	Palabras polisémicas	Sentidos polisémicos
Nombre	101863	15935	44449
Verbo	6277	5252	18770
Adjetivo	16503	4976	14399
Adverbio	3748	733	1832
Total	128391	26896	79450

Tabla 2: Información Polisémica que contiene

WordNet también dispone de una interfaz on-line para poder consultar su base de datos. Nos permite buscar una palabra en inglés junto con con las opciones que queremos que nos muestre de dicha palabra. En la siguiente imagen podremos apreciar un ejemplo:

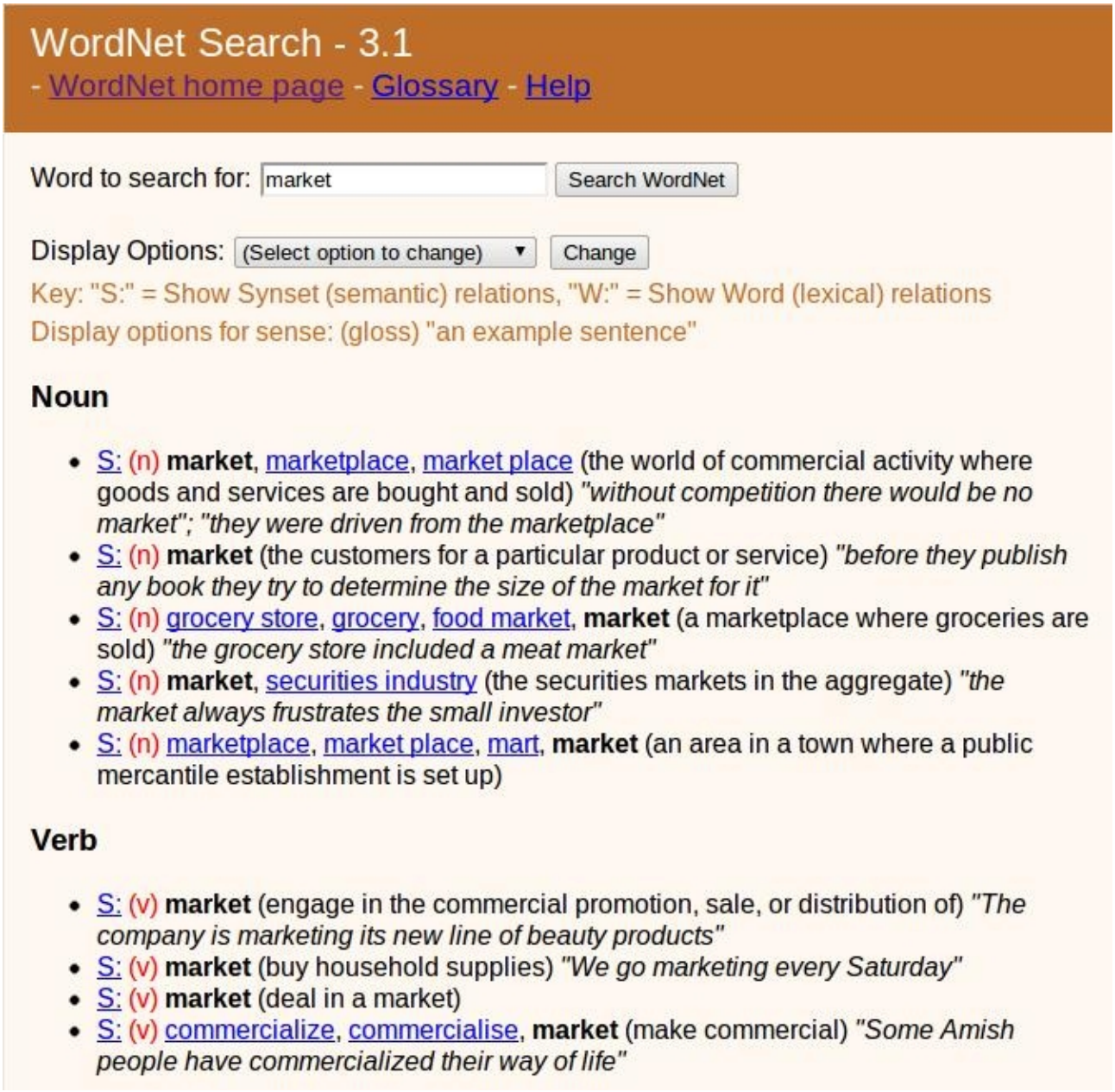


Figura 2: Resultado obtenido de la consulta de una palabra con la herramienta WordNet

Una vez obtenido el resultado, la interfaz nos deja movernos por las diferentes relaciones conectadas con alguno de los sentidos de la palabra que hemos buscado. Se podría decir que es un gráfico dinámico que nos permite acceder a los distintos nodos.

3.2- Word Sense Disambiguation

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.

Los sistemas de WSD han usado tradicionalmente un enfoque superficial. Los enfoques superficiales 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. Por ejemplo, comprobando la presencia de un término en particular en el contexto que rodea a la palabra a desambiguar como en las reglas ``si el verbo *curar* tiene las palabras *jamón* o *embutido* cerca, es probable que sea el sentido de conservación, y si el verbo *curar* tiene las palabras *paciente* o *herida*, es probable que sea el sentido de *sanar*".

Los sistemas de WSD se clasifican generalmente como **supervisados** o **no supervisados**. Sin embargo, hoy en día es difícil establecer una clasificación estricta, ya que existen métodos que utilizan diferentes grados de supervisión. Métodos supervisados son aquellos que utilizan técnicas de aprendizaje automático para aprender clasificadores usando corpus con sentidos-anotados. Por otro lado, aproximaciones como las basados en el uso de grafos derivados de las glosas anotadas con sentidos de WordNet, serán considerados como no supervisados. Además, hay sistemas que combinan ambos enfoques para beneficiarse de sus ventajas. WSD depende de las características del inventario de sentidos utilizados, así como de su granularidad, la cobertura y la riqueza de la información codificada, etc. WSD ha demostrado ser útil para mejorar tareas como en el análisis sintáctico, la recuperación de información (IR), la traducción automática o extracción de la información. A pesar de sus desventajas inherentes, los métodos supervisados basados en corpus obtienen mejores resultados que los métodos no supervisados.

Sin embargo, los métodos no supervisados y, en particular, los métodos basados en grafos, presentan ventajas muy interesantes. No dependen de ningún corpus etiquetado para entrenar. Sin embargo, en comparación, los métodos basados en grafos obtienen mejores resultados cuando se aplican a un conjunto de palabras estrechamente relacionadas, que cuando se aplican a texto corrido. Al aplicar WSD a dominios concretos, los métodos supervisados obtienen peores resultados

en comparación a su rendimiento en textos de dominio general.

3.3- SSI-Dijkstra

El software SSI-Dijkstra es una versión del algoritmo SSI (Structural Semantic Interconnections). SSI-Dijkstra es un algoritmo de desambiguación de sentidos basado en el conocimiento. El algoritmo SSI es muy simple y consiste en un paso de inicialización y en ciertos pasos iterativos (ver algoritmo debajo).

Dada W, una lista ordenada de palabras que deseamos desambiguar, 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 están 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 de sentidos almacenados en I se usa para desambiguar una palabra de la lista P, seleccionando el sentido de palabra de P más próximo al de las palabras ya desambiguadas de la lista I. Una vez seleccionado el sentido, esa palabra la quitamos 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.

```
1   SSI (T: list of terms)
2   for each {t ∈ T} do
3       I[t] = ∅
4       if t is monosemous then
5           I[t] := the only sense of t
6       else
7           P := P ∪ {t}
8       end if
9   end for
10  repeat
11      P' := P
12      for each {t ∈ P} do
13          BestSense := ∅
14          MaxValue := 0
15          for each {sense s of t} do
16              W[s] := 0
17              N[s] := 0
18              for each {sense s' ∈ I} do
19                  w := DijkstraShortestPath(s,s')
20                  if w > 0 then
```

```
21                                          W[s] := W[s] + (1/w)
22                                          N[s] := N[s] + 1
23                                  end if
24                          end for
25                          if N[s] > 0 then
26                                  NewValue := W[s]/N[s]
27                                  if NewValue > MaxValue then
28                                          MaxValue := NewValue
29                                          BestSense := s
30                                  end if
31                          end if
32                  end for
33                  if MaxValue > 0 then
34                  I[t] := BestSense
35                  P := P \ {t}
36                  end if
37          end for
38      until P ≠ P'
39      return (I,P);
```

Inicialmente, la lista de palabras interpretadas I debería incluir los sentidos de las palabras monosémicas de W10[2].

Para calcular la proximidad de un sentido al resto de sentidos de I, se usa el conocimiento que ya tenemos disponible en el grafo construido a partir de los 99,635 nodos (synset) y 636,077 arco, las cuales son relaciones directas sacadas WordNet y eXtended WordNet. La proximidad la calculamos usando el algoritmo Dijkstra, ya que es un algoritmo muy eficiente para calcular cuál es la distancia más corta entre dos nodos cualquiera del grafo.

SSI-Dijkstra tiene unas propiedades muy interesantes. Por ejemplo, es capaz de calcular de forma eficiente cuál es la distancia más corta entre dos nodos cualesquiera del grafo. De hecho, en nuestro caso, el algoritmo compara las distancias entre los sentidos de una palabra con los sentidos de las palabras ya interpretadas en I.

Además, queremos detectar que ésta aproximación es independiente del lenguaje. El mismo grafo se puede usar para diferentes idiomas si existen palabras conectadas a WordNet para ese lenguaje .

2 El algoritmo SSI-Dijkstra general no sabría que hacer en ausencia de monosémicos. Para este caso particular, véase SSI-Dijkstra+, más adelante.

3.4- SSI-Dijkstra+

SSI-Dijkstra+ es una versión extendida de SSI-Dijkstra. Ésta versión trata el caso en el que el conjunto W de palabras a procesar no dispone de palabras monosémicas. Se puede comprobar que, 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-Dijkstra+ obtiene solución cuando los conjuntos de palabras por procesar no tienen monosémicos. Para ello, obtiene la palabra menos ambigua. Ésta palabra es la que menor grado de polisemia posee. Dicho de otro modo, la que menos sentidos asociados tiene a sí misma. Con ella realiza una hipótesis acerca de su significado más correcto. La hipótesis se obtiene a partir del resultado de dos algoritmos diferentes: ASI y FSI.

***ASI**:* Ésta versión halla la distancia mínima acumulada desde cada uno de los sentidos de la palabra seleccionada hasta el resto de sentidos de las palabras de P. El sentido seleccionado de la palabra a tratar será aquel que tenga menor dicha distancia.

***FSI**:* Ésta versión halla la distancia mínima acumulada desde cada uno de los sentidos de la palabra seleccionada hasta el primer sentido de cada palabra de P. El sentido seleccionado de la palabra a tratar será aquel que tenga menor dicha distancia.

Estos dos algoritmos siempre obtienen solución, aún cuando no hay monosémicos. Por tanto, nos aseguramos que el conjunto de palabras interpretadas I, no esté vacío antes del paso iterativo.

En segundo lugar, SSI-Dijkstra+ pretende mejorar la precisión de los resultados obtenidos a partir de la aplicación del algoritmo incluyendo para ello dos nuevos algoritmos durante el paso iterativo: ASP y FSP. Nótese que los dos algoritmos tienen en cuenta el conjunto de palabras interpretadas I para obtener su resultado. La diferencia entre ellos, se explica a continuación:

***ASP**:* Ésta versión halla la distancia mínima acumulada desde un sentido de la palabra por interpretar, hasta cada uno de los sentidos presentes en I y el resto de sentidos de las palabras de P.

El sentido seleccionado de la palabra a tratar será aquel que tenga menor dicha distancia.

FSP: Esta versión halla la distancia mínima acumulada desde un sentido de la palabra por interpretar hasta cada uno de los sentidos presentes en I y sólo el primer sentido de cada palabra de P. El sentido seleccionado de la palabra a tratar será aquel que tenga menor dicha distancia.

La inserción de P o no, a la hora de desambiguar palabras, es una cuestión difícil, que ha sido estudiada de forma empírica (Laparra et al., 2009). Al parecer, los verbos se desambiguaban bastante mejor utilizando el FSP; mientras que el resto de palabras se desambiguaban con un resultado de precisión mejor sin tener en cuenta el conjunto de palabras sin desambiguar P.

En lo que respecta a este proyecto, se ha decidido utilizar FSI para la parte de inicialización (sólo si el conjunto I está vacío después de la inicialización inicial). Para la parte iterativa, se ha decidido usar FSP (sólo para verbos) o el propio algoritmo SSI-Dijkstra (para nombres, adjetivos y adverbios). Esta opción pretende optimizar tiempo y esfuerzo. Aunque otras decisiones son igualmente válidas.

3.5- UKB

UKB[3] es un software desarrollado por el grupo IXA[4] de la Universidad del País Vasco para trabajar en el campo de la desambiguación de palabras. Codificado en C++, es una librería que permita la creación, lectura y procesamiento de grafos a partir de ficheros, los cuales contienen la información necesaria para la desambiguación de palabras.

Teniendo en cuenta lo anterior, UKB ha sido uno de los pilares sobre los que se ha sustentado este proyecto debido a sus facilidades para trabajar con bases de conocimiento en forma de grafos. UKB proporciona facilidades para procesar ficheros que almacenan la estructura de un grafo. Ésta ha sido una de las razones más importantes para elegir este software en éste proyecto. Se debe tener en cuenta también que el propio UKB incluye diferentes ficheros con diferentes versiones de WordNet para trabajar con él. Así mismo, tiene también un diccionario para poder enlazar cada palabra a sus sentidos asociados (y las utilidades de uso de dicho diccionario).

Es importante también señalar que, UKB proporcionaba casi todas las herramientas necesarias para este proyecto, pero no todas. Como uno de los objetivos del proyecto era intentar reutilizar al máximo el código propio de UKB, se han tenido que introducir utilidades y métodos que antes no tenía. Ésto es algo importante a tener en cuenta, ya que este proyecto, sin UKB, no funciona, pero tampoco lo hace con una versión errónea del mismo (actualmente 0.1.5r1, aunque faltaría actualizar la versión).

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

4 http://ixa.si.ehu.es/Ixa

4- Elección tecnológica

Parte de la elección de la tecnología ha estado bastante clara, ya que necesitábamos unas herramientas concretas para la elaboración del proyecto. Teniendo eso en cuenta, aún tenemos cierta libertad a la hora de realizarlo.

4.1-Pre-requisitos

Antes de explicar la tecnología utilizada, se expondrán las herramientas fundamentales para el desarrollo del programa.

- El proyecto entero está hecho en C++, por lo tanto, se necesitaba una máquina capaz de compilar dicho lenguaje.

- Utilización del software UKB, ya que es esencial para la elaboración del mismo. Utilizamos varias clases de dicho software, además de estar el proyecto integrado en él.

- También son necesarias la librería BoostGraph de C++, ya que UKB y nuestro proyecto la necesita.

4.2-Desarrollo

4.2.1-Sistema operativo

El sistema operativo elegido ha sido Linux Ubuntu 11.10. Se ha hecho ésta elección por recomendación del director y por que el alumno mostró disposición por aprender y familiarizarse con él. Otra de las razones ha sido por la facilidad que ofrece Linux Ubuntu de trabajar con C++ frente a otros sistemas.

4.2.2-Entorno de desarrollo

No ha sido necesario ningún entorno (como por ejemplo Eclipse). Se ha optado por utilizar el editor de texto gedit de Linux . Dicho editor era lo suficientemente potente para cubrir las necesidades del alumno.

4.3-Documentación

Es necesario decidir la tecnología utilizada para realizar la documentación del proyecto. Se ha elegido el paquete ofimático Libre Office, frente a Microsoft Word, por el sistema operativo usado. Tampoco se ha usado el OpenOffice, ya que el sistema operativo traía el Libre Office por defecto, y hemos considerado que ambos eran muy similares (ya que la base del Libre Office es el OpenOffiece).

4.3.1.- LibreOffice Writer. Memoria

LibreOffice.org Writer es un procesador de texto multiplataforma, que forma parte del conjunto de aplicaciones de la suite ofimática LibreOffice.org. Además de otros formatos estándares ampliamente utilizados de documentos, puede abrir y grabar el formato propietario .doc de Microsoft Word casi en su totalidad. El formato nativo para exportar documentos es XML. También puede exportar a ficheros PDF nativamente sin usar programas intermedios.

4.3.2.- LibreOffice Impress. Transparencias

LibreOffice.org Impress es un programa de presentación similar a Microsoft PowerPoint. Es parte de la suite de oficina de LibreOffice.org desarrollada por la fundación The Document Foundation. Puede exportar presentaciones, tales como archivos SWF de Adobe Flash, permitiendo que sean ejecutados en cualquier computadora con Adobe Flash Player instalado. También incluye la capacidad de crear archivos PDF.

4.3.3.- Gantt Project

Gantt Project es una herramienta de escritorio multiplataforma para la planificación y gestión de proyectos. Funciona en Windows, Linux y MacOSX, y es gratis y de código abierto. Fundamentalmente, se utiliza para hacer diagramas de Gantt aunque permite hacer también diagramas PERT y tablas de asignación de recursos, además de otras cosas útiles. Permite además exportar sus diagramas en formato de imagen, lo cuál será bastante interesante a la hora de integrarlos en esta memoria.

5- Captura de requisitos

Como hemos comentado anteriormente, SSI-Dijkstra es un algoritmo para la desambiguación de palabras que usa UKB, ya que desde el principio se pensó aprovecharlo al máximo. UKB es una herramienta usada en la desambiguación de palabras por métodos basados en grafos, y que es fundamental para el desarrollo del proyecto. Está escrita en C++ y utiliza las librerías BoostGraph.

Es importante que el SSI-Dijkstra sea capaz de entender, procesar y trabajar con la misma entrada que UKB para que diera una salida de formato idéntica a la de UKB . Para cumplir con este requisito, tuvimos que integrar nuevos tipos de palabras al SSI-Dijkstra, ya que las nuevas versiones del UKB las tenían:

-Palabras que deben ser desambiguadas pero que no tenían que ayudar en la desambiguación de las demás (las de tipo *"cw_tgtword_nopv"*).

-Palabras que deben ser desambiguadas y ayudar a desambiguar al resto, pero que no tenían que mostrarse en el resultado *(las del tipo "cw_ctxword").*

Aparte de los puntos principales a tratar, se decidió introducir funcionalidades que podrían ayudar en la eficiencia y eficacia del programa:

Era necesario algo que fuera capaz de ordenar las palabras del conjunto, de forma que al utilizar SSI-Dijkstra+, las primeras palabras fueran "las menos polisémicas"[5]. El objetivo es intentar maximizar la eficacia del programa desambiguando primero las menos ambiguas, haciendo que el conjunto de palabras quede ordenado de forma creciente en cuanto al grado de polisemia. Además, aprovechando las propiedades del formato de la entrada de UKB, se quiso dar la posibilidad a que el usuario pudiera definir el orden en el que las palabras eran procesadas y surgió la idea de crear una ordenación explícita.

5 Se define como palabra "menos polisémica" aquella que tiene un menor número de sentidos asociados. A este método de ordenación se le refiere también como ordenación por grado de polisemia .

Era fundamental que el programa pudiera trabajar con SSI-Dijkstra o SSI-Dijkstra+ indistintamente, según decisión propia del usuario. De esa forma, podemos obtener resultados de fiabilidad para ambos algoritmos y analizar en qué casos es mejor uno que otro. También podemos obtener respuestas de SSI-Dijkstra+ en casos en los que SSI-Dijkstra no nos las daría. Se observó un método para obtener respuestas de forma más rápida en los sucesivos cálculos de las distancias entre nodos del grafo. Este método lo denominaremos método con “optimización”, mientras que la forma normal de hacerlo la denominaremos método sin “optimización” (los detalles técnicos vienen descritos en el apartado de implementación).

Es interesante que el usuario pudiera utilizar grafos definidos por él mismo, de manera que pudiera utilizar las herramientas de UKB para crear el grafo siguiendo un patrón definido por el propio usuario. Por lo que también era requisito del proyecto que el usuario pudiera decidir fácilmente el grafo que quería utilizar.

Nos dimos cuenta que el algoritmo SSI-Dijkstra era más ineficiente que el SSI-Dijkstra+, ya que éste último usaba un algoritmo especial para tratar los verbos, que hacía que ganase más precisión. Para solucionar ésto, se decidió incluir ese algoritmo en el SSI-Dijkstra para que ambos tuviesen el mismo rango de aciertos.

Para mejorar aún más la eficiencia, decidimos ordenar las palabras que iban a tener que ser desambiguadas. Ésto se decidió puesto que uno de los tipos de palabras, las que no ayudaban en la desambiguación pero que tenían que ser desambiguadas, debían colocarse al final para ganar la eficiencia anteriormente comentada.

Tras la implementación del SSI-Dijkstra y SSI-Dijkstra+, con algunas de las opciones comentadas anteriormente, y los cambios en el UKB, se realizaron una serie de pruebas en profundidad. En dichas pruebas se pudo observar que el programa no funcionaba de una forma correcta, ya que los resultados obtenidos diferían de los esperados. En el siguiente ejemplo podemos apreciar claramente el error en cuestión:

- **Ejemplo:**

  ctx_08264897-n(08240169n)
  08264897-n#n#wf0#2 band#n#wf2#1 people#n#wf4#1 00713167-v#v#wf5#2
  00088303-r#r#wf6#2 activity#n#wf9#1

- **Resultado erróneo:**

  ctx_08264897-n(08240169n)wf5 00713167-v !! 00713167-v
  ctx_08264897-n(08240169n)wf6 00088303-r !! 00088303-r
  ctx_08264897-n(08240169n) wf9 14006945-n !! activity

- **Resultado correcto:**

  ctx_08264897-n(08240169n) wf2 08240169-n !! band
  ctx_08264897-n(08240169n) wf4 07942152-n !! people
  ctx_08264897-n(08240169n) wf9 14006945-n !! activity

Como podemos apreciar en el primer resultado, solo obtenemos una de las palabras que esperamos (activity), ya que el resto son palabras que ya estaban desambiguadas y no tenían que mostrarse. En el segundo ejemplo se puede apreciar que desambigua de forma correcta las tres palabras que nos interesaba desambiguar. Por tanto el objetivo principal es el siguiente:

-Identificar el error por el cual no es capaz de desambiguar de forma correcta el contexto.

5.1-Modelo de Casos de uso

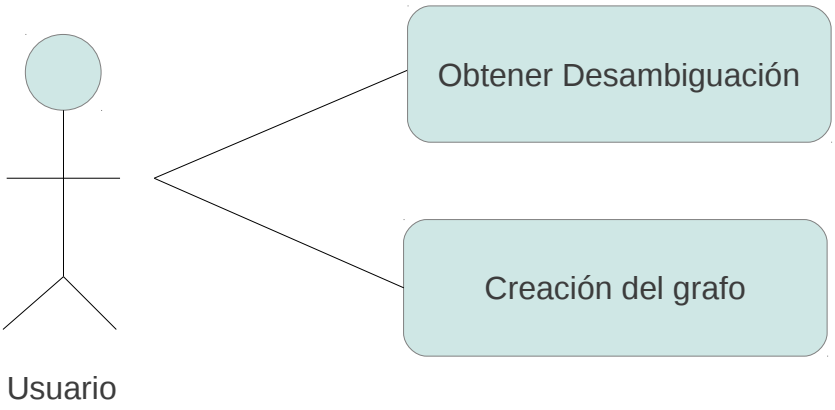


Figura 3: Caso de Uso del usuario

Obtener desambiguación

***Actores*:** Usuario

Descripción: Ésta funcionalidad permite al usuario, dado un contexto, obtener la desambiguación del mismo mediante el algoritmo elegido, SSID-Dijkstra+ o SSID-Dijkstra.

Caso normal de eventos:

1- El usuario hace la llamada especificando el número de opciones (que explicaremos más adelante) y el fichero con los contextos a desambiguar.

2- Se muestra por la salida estándar las palabras desambiguadas. El resultado puede ser diferente en función de las opciones introducidas

Flujos alternativos:

1.1- Si el usuario introduce mal alguna de las opciones, se muestra un error indicándolo.

1.2- El usuario puede indicar si quiere introducir su propio grafo y diccionario para hacer la desambiguación.

1.3- Si los contextos, del fichero que le pasamos, no están bien construidos (ya sea porque contiene alguna palabra en mayúsculas, falta algún campo...), obtendríamos un mensaje indicándonos el error.

Como ya hemos comentado anteriormente, podemos utilizar diferentes opciones que pueden modificar la forma en la que el programa desambigua las palabras o los datos obtenidos tras la desambiguación. A continuación, nos dispondremos a explicar las diferentes opciones disponibles, junto a su uso dentro del algoritmo:

a) Generales:

--**dict_weight** : Como en UKB, permite indicar que se usarán los pesos presentes en el diccionario.
-C o --only_ctx_words : Como en UKB, permite obligar al programa a que trabaje sólo con las palabras de un contexto y no con todo el diccionario.
--**pretty :** Indica que los datos se quieren ordenar por id antes de sacarlos por pantalla.
--**show_all_results:** Muestra las palabras que ya estaban desambiguadas y las de un tipo especial que de forma normal tampoco se mostrarían. Esta opción la incluimos ya que el UKB no la tenía.

b) Ordenación (con clara influencia en la desambiguación):

--**nexp:** Permite indicar que no se seguirá ningún criterio de ordenación concreto (los datos se procesarán según llegan). Ésta es la opción que se usa por defecto si no se ha introducido ninguna otra.
--**poly:** Permite indicar que se ordenarán las palabras por grado de polisemia (menos polisémicas primero).
--**expl:** Permite indicar que se ordenarán las palabras según orden explícito (indicado en el fichero de contextos).

c) Algorítmica

--**ssid:** Indica que se utilizará SSID normal en lugar de SSID+.
--**no-opt:** Indica que se utilizará la invocación de dijkstra_shortest_paths sin "optimización" en lugar de la optimizada.

Creación del grafo

Actores: Usuario

Descripción: Ésta funcionalidad permite al usuario crear un grafo

Caso normal de eventos

1-El usuario hace la llamada indicando el nombre del archivo en el cual se guardará el grafo y los archivos necesarios para poder crearlo.

2-Se crea el archivo, que está listo para usarse.

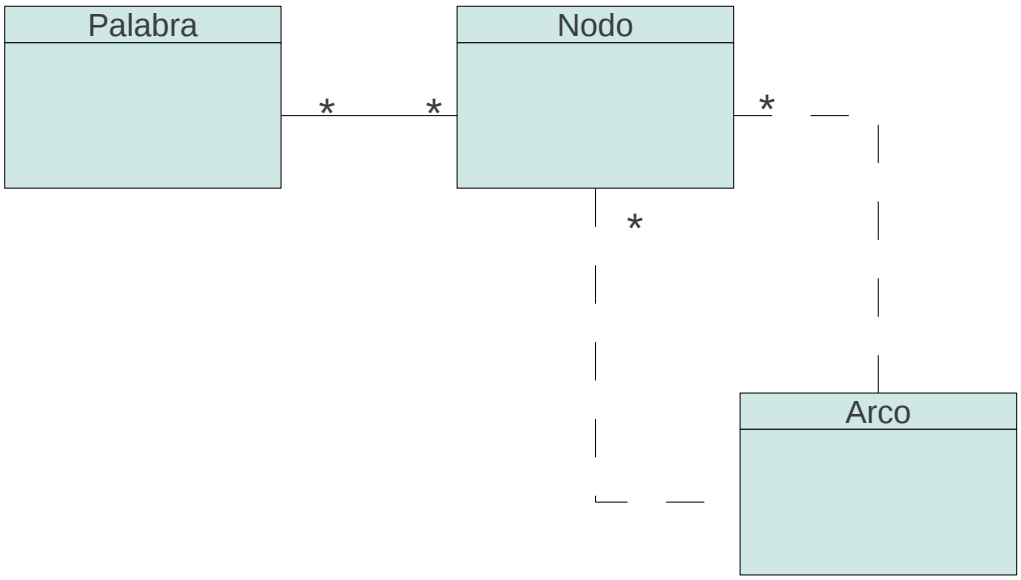
Flujos alternativos

1.1-Si los archivos necesarios para la creación del grafo no existe, o la ruta no está bien especificada, nos mostraría un error indicando dicho suceso.

1.2-Si los archivos necesarios para la creación del grafo no tienen una correcta estructura, nos mostraría un error.

5.2-Modelo de Dominio

A continuación, mostraremos un Modelo de Datos referente a nuestro proyecto:



Ahora pasaré ha explicar cada una de las estructuras anteriores:

a) Palabra

La palabra contiene la información necesaria para poder desambiguarse, teniendo la siguiente estructura:

lemma#pos#id#d#w

- lemma: contiene la lema de la palabra.
- pos: nos dice de que tipo es la palabra (nombre, verbo, adjetivo o adverbio).
- id: la id de la palabra.
- d: contiene el número de control que nos indica qué acciones tenemos que llevar a cabo con ella.
- W: contiene el peso de la palabra.

En el siguiente ejemplo podemos apreciar con más claridad dicha estructura:

together#r#wf12#1

b) Nodo

El nodo contiene la información necesaria para poder hacer las relaciones y desambiguar la palabra. Su estructura es la siguiente:

- idx: es un número único que representa al nodo.
- pos: nos dice de que tipo es la nodo (nombre, verbo, adjetivo o adverbio).

En el siguiente ejemplo podemos apreciar con más claridad dicha estructura:

00001740-n

c) Arco

El arco representa la relación entre dos vértices del KB . Su estructura es la siguiente:

- u: el vértice origen de la relación.
- v: el vértice destino de la relación.
- d: para saber si la relación es dirigida.
- s: el origen de la relación.
- w: el peso de la relación.

En el siguiente ejemplo podemos apreciar con más claridad dicha estructura:

u:00023741-n v:02223033-a d:1 w:0.3 s:wn30

Como podemos observar en el modelo de datos, cada palabra puede tener más de un nodo asignado, y a su vez, cada nodo puede tener más de una palabra asignada. Ésto sucede con palabras que son sinónimas entre si, por tanto, comparten el mismo significado. Cada nodo, a su vez, está relacionado con más nodos. Por ejemplo, la palabra partido estará relacionada con: jugadores, portería, balón, elecciones, político...Ésto sucede, ya que dicha palabra puede referirse a un partido de algún deporte o a un partido político. Con esta información, podemos hacernos una idea de la estructura interna del grafo.

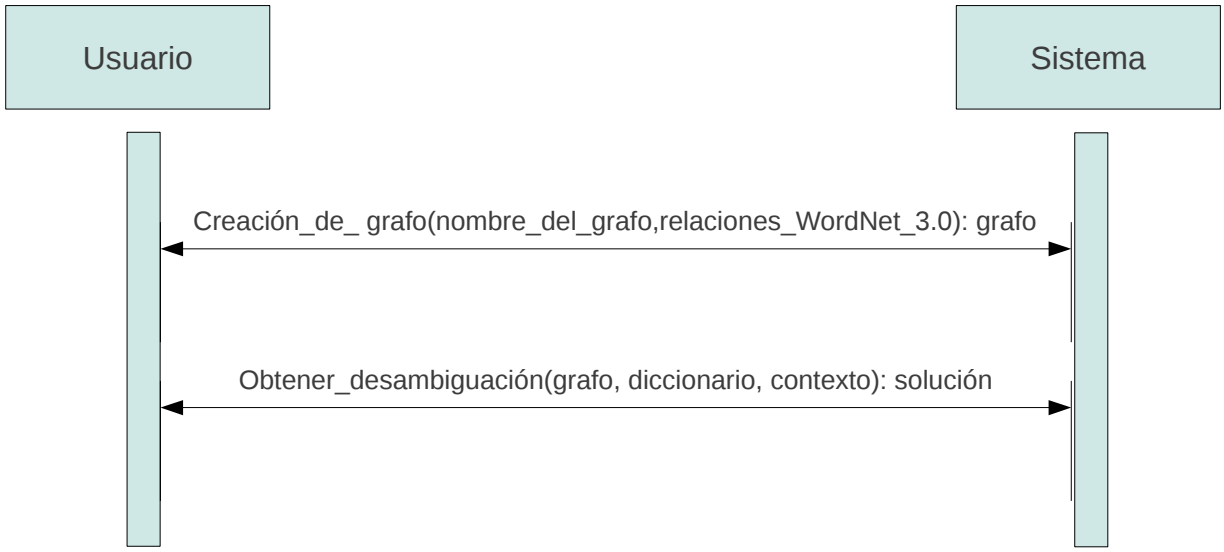
6- Análisis

Como hemos comentado anteriormente, el programa no funciona de una forma correcta, ya que no nos muestra por la salida estándar todas las palabras que tiene que desambiguar, ya que también nos muestra las que ya están desambiguadas desde un principio. Se pudo observar que las palabras que se muestran están bien desambiguadas por lo que se piensa que el algoritmo desambigua correctamente. Se barajan diferentes opciones por la cuales el programa pueda estar actuando de esa manera:

1) La función con la cual mostramos las palabras por la salida estándar no funciona bien, por tanto no muestra todo lo que debería.
2) Las palabras desambiguadas no se introducen en la estructura correspondiente, que es la que le pasamos a la función de visualización para que las muestre.
3) Las palabras no se crean de una forma correcta, ya que puede que alguno de los parámetros no se introduzcan bien.
4) Durante la ejecución del algoritmo, puede que se modifiqué algún parámetro y se introduzca uno incorrecto en su lugar.

Tras varias pruebas, se detectó que el error estaba en el parámetro que indicaba de qué tipo era la palabra. Cuando la palabra era creada, no se introducía correctamente el tipo, ya que se usaba el mismo para todas, ya sean palabras que debían ser desambiguadas o las que ya estaban desambiguadas desde el principio. Ésto también sucedía cuando la palabra era introducida en la estructura que se le pasaba al algoritmo de visualización. El constructor creado por el alumno anterior, asignaba incorrectamente el tipo de la palabra que era, por lo que se tenía que modificar para que lo asignase de forma correcta. También se tuvo que cambiar el operador “=” de Csentence ya que cuando se igualaba una palabra a otra**,** no se asignaba correctamente el tipo de la palabra anterior a la nueva.

6.1-Casos de Uso



Contratos:

Nombre: Creación del grafo.

Responsabilidades: Que existan los archivos necesarios para la creación del grafo (realaciones_WordNet_3.0), que estén bien construidos y el directorio esté bien indicado.

Pre-condición:

Post-condición: Creación del grafo con el nombre indicado por el usuario.

Salida:

Nombre: Obtener desambiguación.

Responsabilidades: Que los archivos necesarios para la desambiguación (diccionario, fichero de contextos y grafo) existan, que estén bien construidos y el directorio esté bien indicado. En caso de introducir alguna opción, que esté bien escrita.

Pre-condición:

Post-condición:

Salida:Visualización de las palabras desambiguadas por la salida estándar.

Contrato de la capa de presentación

Nombre:Visualización de las palabras desambiguadas.

Responsabilidades: Introducirle una estructura, con las palabras a visualizar, válida para que sea capaz de leerla.

Pre-condición: Las palabras introducidas se encuentras ya desambiguadas.

Post-condición:

Salida: Muestra por la salida estándar el resultado de la desambiguación, que puede variar en función de las opciones introducidas.

7- Diseño

7.1-Obtener distancia

Descripción: Es un algoritmo que dados dos Kb_vertex_t (nodos del grafo en UKB) obtiene la distancia mínima entre ellos en el KbGraph (grafo en UKB).

Pseudocódigo:

```
obtener_distancia(vertice1, vertice2, grafo) {
        dijkstra_shortest_paths(grafo, vertice1, predecessor_map(padres).distance_map(distancias));
        N = nodos(grafo);
        mi_iterador = apuntador(distancias[0]);
        mientras mi_iterador != apuntador(distancias[N]) repetir
                si mi_iterador = apuntador(vertice2) entonces
                        distancia = distancias[mi_iterador];
                        salir;
                finsi
                mi_iterador++;
        finmientras
        devolver distancia;
}
```

7.2- Obtener distancia más rápido

Descripción: Es un algoritmo que, dados dos Kb_vertex_t (nodos del grafo en UKB), obtiene la distancia mínima entre ellos en el KbGraph (grafo en UKB). El cuarto parámetro es un Kb_vertex_t para controlar que se calcule el dijkstra_shortest_paths, sólo si éste último parámetro es diferente del primero[6].

Pseudocódigo:

```
obtener_distancia_mas_rapido(vertice1, vertice2, grafo, vertice_anterior) {
        si vertice1 != vertice_anterior entonces
                dijkstra_shortest_paths(grafo,vertice1,
                predecessor_map(padres).distance_map(distancias));
        finsi
        N = nodos(grafo);
        mi_iterador = apuntador(distancias[0]);
        mientras mi_iterador != apuntador(distancias[N]) repetir
                si mi_iterador = apuntador(vertice2) entonces
                        distancia = distancias[mi_iterador];
                        salir;
                finsi
                mi_iterador++;
        finmientras
        devolver distancia;
}
```

6 Los detalles relacionados con esto, pueden encontrarse en el apartado de Implementación .

7.3-Mostrar Camino

Descripción: Es un algoritmo que, dados dos Kb_vertex_t (nodos del grafo de UKB), obtiene el camino mínimo entre ellos en el KbGraph (grafo en UKB) y lo muestra por la salida estándar.[7]

Pseudocódigo:

```
mostrar_camino(vertice1, vertice2, grafo) {
        N = nodos(grafo);
        mi_iterador = apuntador(distancias[0]);
        mientras mi_iterador != apuntador(distancias[N]) {
                si mi_iterador = apuntador(vertice2) entonces
                        vector.añadir(vertice2);
                        aux_pointer = padres[aux_pointer]
                        mientras iterador_auxiliar != vertice1 repetir
                                si aux_pointer == iterador_auxiliar
                                entonces
                                        vector.añadir(aux_pointer)
                                finsi;
                        finmientras;
                        salir;
                finsi;
                mi_iterador++;
        finmientras;
        imprimir(vector);
}
```

7 No es una parte del proyecto en sí, pero es una función fantástica para comprobar la validez de algunos resultados.

7.4-SSI-Dijsktra

Descripción: Es una función que, recibiendo un contexto de entrada, genera una nueva estructura de datos de contexto con las palabras del contexto de entrada desambiguadas.

Pseudocódigo:

```
SSI_Dijkstra (contexto, grafo) {
    // Inicialización
    para cada palabra ∈ contexto
        caso(palabra.significados())
            caso 0: // Algo extraño ha pasado
                romper;
            caso 1: // Palabra monosémica
                I.añadir(palabra.significados()[0])
                solucion.añadir(palabra)
                romper;
            defecto: // Palabra polisémica
                P.añadir(palabra)
                romper;
        fin-caso
    fin-repetir

    // Paso iterativo
    si I ≠ ∅ entonces
        mientras P ≠ ∅
            vector_P = P[0].significados()
            total = 0
            synsets = 0
            para cada significadoP ∈ P
                para cada significado'I ∈ I
                    w = obtener_distancia(significadoP, significado'I, grafo);
```

```
                    si w > 0
                        total = total + (1/w)
                        synsets++
                    fin-si
                fin-para
                si synsets > 0
                    v = total/synsets
                    si v > max
                        max = v
                        best = significado'I
                    fin-si
                fin-si
            fin-para
            si max > 0
                I.añadir(best)
                solucion.añadir(new Palabra(best))
                P.borrar(P[0])
            fin-si
        fin-mientras
    fin-si
    devolver solucion
}
```

7.5-SSI_Dijkstra+

Descripción: Es una función que, recibiendo un contexto de entrada, genera una nueva estructura de datos de contexto con las palabras del contexto de entrada desambiguadas.

Pseudocódigo:

```
SSI_Dijkstra+ (contexto, grafo) {
    // Inicialización: parte 1
    para cada palabra ∈ contexto
        caso(palabra.significados())
            caso 0: // Algo extraño ha pasado
                romper;
            caso 1: // Palabra monosémica
                I.añadir(palabra.significados()[0])
                solucion.añadir(palabra)
                romper;
            defecto: // Palabra polisémica
                P.añadir(palabra)
                romper;
        fin-caso
    fin-repetir
    // Inicialización: parte 2 (FSI)
    si I ≠ ∅ entonces
        palabraPtoI = P[0].significados()
        i = 1
        mientras i ≠ P.tamaño()
            total = 0
            synsets = 0
            palabraP = P[i]
            significadoPtoI = palabraPtoI.significados()[0]
            para cada significadoP ∈ palabraP
```

```
                    w = obtener_distancia(significadoP, significadoPtoI, grafo)
                    si w > 0
                        total = total + (1/w)
                        synsets++
                    fin-si
                fin-para
                si synsets > 0
                    v = total/synsets
                    si v > max
                        max = v
                        best = significadoPtoI
                    fin-si
                fin-si
            fin-mientras
            si max > 0
                I.añadir(best)
                solucion.añadir(new Palabra(best))
                P.borrar(P[0])
            fin-si
    fin-si
    // Paso iterativo
    si I ≠ ∅ entonces
            mientras P ≠ ∅
                vector_P = P[0].significados()
                total = 0
                synsets = 0
                para cada significadoP ∈ P
                    para cada significado'I ∈ I
                        w = obtener_distancia(significadoP, significado'I, grafo)
                        si w > 0
                            total = total + (1/w)
                            synsets++
```

```
                    fin-si
                fin-para
                si P[0].pos() = 'v' // FSP
                    i = 1
                    mientras i ≠ P.tamaño()
                        palabra = P[i]
                        significadoP = palabra.significados()[0]
                        w = obtener_distancia(significadoP,
                        significado'I, grafo)
                        si w > 0
                            total = total + (1/w)
                            synsets++
                        fin-si
                    fin-mientras
                fin-si // Fin FSP
                si synsets > 0
                    v = total/synsets
                    si v > max
                        max = v
                        best = significado'I
                    fin-si
                fin-si
            fin-para
            si max > 0
                I.añadir(best)
                solucion.añadir(new Palabra(best))
                P.borrar(P[0])
            fin-si
        fin-mientras
    fin-si
    devolver solucion
}
```

7.6- Ordenar por polisemia

Descripción: Es una función que compara el número de significados asociados de diferentes palabras de un contexto, y devuelve si la primera palabra tiene menos significados asociados que la segunda.

Pseudocódigo:

```
ordenar_polisemia (palabra1, palabra2) {
        vector1 = palabra1.significados()
        vector2 = palabra2.significados()
        devolver vector1.longitud() < vector2.longitud()
}
```

7.7-Ordenar de forma explícita

Descripción: Es una función que compara los pesos de diferentes palabras de un contexto y devuelve si el primer peso es menor que el segundo .

Pseudocódigo:

```
ordenar_explícito (palabra1, palabra2) {
        peso1 = palabra1.peso()
        peso2 = palabra2.peso()
        devolver peso1 < peso2
}
```

7.8-Ordenar por ID

Descripción: Es una función que compara los ids de diferentes palabras de un contexto y devuelve si el primer id es “menor” (alfabéticamente hablando, es decir, si va antes) que el segundo .

Pseudocódigo:

```
ordenar_ids (palabra1, palabra2) {
        id1 = palabra1.id()
        id2 = palabra2.id()
        devolver id1 < id2
}
```

7.9-Ordenar palabras

Descripción: Es una función que ordena un contexto en función de una opción especificada. La opción es un parámetro de la función.

Pseudocódigo:

```
ordenar_palabras(contexto, opcion) {
        caso(opcion)
                caso 0: // Sin ordenar
                        romper;
                caso 1: // Ordenar por polisemia
                        sort(contexto.principio, contexto.final, ordenar_polisemia)
                        romper;
                caso 2: // Orden explícitos
                        sort(contexto.principio, contexto.final, ordenar_explícito)
                defecto: // Imposible
                        romper;
        fin-caso
```

}

7.10-Ordenación por id

Descripción: Es una función que ordena un contexto en función de su id utilizando el sort de C++ (la función auxiliar es la que queda más arriba). Por ejemplo, si los ids fueran w1, w2 y w10 devolvería un contexto ordenado de esta forma: w1, w10, w2.

Pseudocódigo:

```
ordenar_id (contexto) {
        sort(contexto.principio, contexto.final, ordenar_ids)
}
```

7.11-Programa principal (main)

Descripción: Es el programa principal del proyecto. Básicamente, lo que hace es comprobar la corrección del proyecto, ya que lo que hace es recoger datos (leer contextos), procesarlos (llamando a la función de desambiguación correspondiente) e imprimir datos (resultados de la desambiguación).

Pseudocódigo:

```
main (...) {
        // Configuración de las opciones del program_options
        // Leer grafo + diccionario...
        // Leer del fichero de contextos correspondiente
        mientras existan contextos por leer repetir
                estructura_UKB = leer_contexto();
                estructura_UKB = calcular_SSID(estructura_UKB);
                imprimir(estructura_UKB);
        fin-repetir
```

}

7.12-Mostrar resultado

Descripción: Algoritmo que muestra por pantalla las palabras desambiguadas. Mostrará más o menos resultados si hemos añadido la opción en la llamada o no. Ésto se realizó en versiones posteriores, con la introducción del nuevo tipo de palabra

Mostrar_el _resultado(opción){

iterador_del_principio = principio del vector;
iterador_que_apunta_al_final = final del vector;

mientras que el iterador no haya recorrido todo el vector **repetir**

si el tamaño de la palabra es 0 **entonces**
pasa al siguiente elemento
s**i** no hemos hemos introducido la opción cuando realizamos la llamada **entonces**
comprobamos las siguientes condiciones
si es del tipo cw_ctxword **entonces** pasamos al siguiente elemento
si la palabra no tiene que ser desambiguada **entonces** pasamos al siguiente elemento

si la palabra es monosémica **entonces** no se muestra

se muestran por pantalla la palabra.
}

7.13-Diferenciar palabras

Descripción: Algoritmo que controla el nuevo tipo de palabra " cw_tgtword_nopv ". En vez de introducirla en la lista de interperted y la de soluciones, solo se añade a la lista de soluciones. Esto sucede, ya que las palabras de ése tipo no tienen que ayudar en la desambiguación. Este algoritmo se encuentra incluido en SSI_Dijkstra y SSI_Dijkstra+, ya que se tuvo que añadir en versiones posteriores, con la introducción de los nuevos tipos.

si la palabra no es del tipo cw_tgtword_nopv **entonces**
    introducimos la palabra dentro del vector
    introducimos la palabra dentro del vector de soluciones
sino
    introducimos la palabra dentro del vector de soluciones

7.14-Ordenar palabras

Descripción: Algoritmo de que coloca las palabras " cw_tgtword_nopv " al final del vector para que la desambiguación se haga de una forma más eficiente. Este algoritmo se encuentra incluido en SSI_Dijkstra y SSI_Dijkstra+, pero no está incluido en el punto anterior, ya que se introdujo más tarde, como una optimización.

    mientras aún queden elemento sin mirar en el vector h**acer**
        si la palabra es de tipo cw_tgtword_nopv() **entonces**
            la ponemos al final de la lista
fin mientras

8- Implementación

Como hemos comentado anteriormente, se intentó reutilizar todo lo que se pudiese del UKB, pero se tuvo que crear varias cosas, ya que el UKB no disponía de ellas. En este apartado, explicaremos un poco más las modificaciones que se hicieron y se han hecho en el UKB. También se comentarán las estructuras utilizadas para un mayor entendimiento de las mismas y de los cambios realizados.

8.1- BoostGraph

Boost es un conjunto de librerías de software libre y revisión por pares preparadas para extender las capacidades del lenguaje de programación C++. Boots contiene una gran cantidad de librerías: procesamiento de cadenas de texto, contenedores... En nuestro caso, el que nos interesa es el BoostGraph, que contiene componentes y algoritmos genéricos de grafos.

8.2-Estructura de ficheros de la versión 1.5

Common: contiene las variables globales, constantes.....

Compile_kb.cc: crea una máquina de serialización, que es dependiente de un grafo de KB., y a su vez, es utilizado por el resto de los programas.

DisamBraph: contiene diferentes operaciones para la desambiguación del grafo.

FileElemm: contiene diferentes operaciones para tratar los archivos de entrada.

GlobalVars: contiene variables globales del programa.

KbGraph: contiene las diferentes operaciones para la creación del grafo, como leer y escribir, desde diferentes tipos de archivo.

Prank.h: contiene las funciones para el tratamiento de las 'PageRank'

UKB_ppv: obtiene el rango de los conceptos del KB.

UKB_wsd: obtiene la desambiguación de las palabras mediante el método de Personalized PageRank.

Wdict: diferentes operaciones para el tratamiento del diccionario.

Csentence.h: contiene las diferentes estructuras que hemos usado para nuestro programa. Entre ellas destacamos la estructura de la palabra y la que contiene las palabras del contexto. A continuación explicaremos con más detalle estas dos estructuras:

El Csentence es una de las estructuras necesarias para el funcionamiento del SSI-Dijkstra, ya que guarda los datos asociados al contexto, siendo ésos datos: una id del contexto y un vector de Cword (que explicaremos a continuación).

Cword es un tipo de estructura que guarda toda la información relacionada con la palabra. Esta estructura contiene más información de la que realmente necesitamos, que despreciaremos, pero entre ella está la información necesaria para nosotros.

Se decidió usar dicha estructura, ya que el propio UKB tenía implementadas diferentes funciones de lectura y escritura, con las que nos facilitarían el trabajo. También se tuvo que modificar algunas cosas.

Ahora pasaremos a comentar la información que contiene el Cword que nos resulta útil a nosotros:

a) Un string en el que guardaremos la palabra a desambiguar.
b) Un carácter donde guardaremos el PoS de la palabra.
c) Una lista de strings asociados a los sentidos de la palabra.
d) Un atributo booleano que indica si la palabra esta desambiguada o no.
e) Un peso propio de la palabra.
f) Un identificador propio de la palabra que nos permite identificar la palabra dentro de un contexto con un identificador cualquiera.

Csentence.cc: contiene los constructores de las estructuras del csentence.h y diferentes funciones para visualizar los datos por la salida entándar.

SSI-Dijkstra: contiene las diferentes operaciones para poder utilizar los algoritmos de SSI-Dijkstra y SSI-Dijkstra+.

8.3-SSI-Dijsktra 1.5: Cambios realizados

En ésta parte, no hemos creado nada nuevo, ya que sólo ha sido necesario modificar ciertas partes para que el programa funcionase de la forma correcta.

- Modificación del constructor de palabras realizado por el alumno anterior. El actual constructor creaba todas los tipos de palabras de la misma forma sin distinguir si eran palabras ya desambiguadas o palabras que tenían que ser desambiguadas. Al hacer éso, tampoco especificaba de forma correcta qué palabras tenían que ser desambiguadas y cuáles no, ya que el atributo que lo indicaba era el mismo para todas. Para solucionarlo, se comprobaba qué tipo de palabra era y se asignaba el valor correcto al atributo.

- Modificación del operador "=" de CSentence. Cuando queríamos guardar una palabra en otra variable, usábamos dicho método, pero asignaba de forma correcta los demás atributos salvo el del tipo de palabra que era, por lo que la palabra no se creaba de forma correcta.

- Modificación en la forma de crear las palabras. A la hora de crear una palabra, se introducía manualmente el tipo de palabra que era, por lo que decidimos coger directamente el valor del atributo para que el algoritmo sea más reutilizable.

8.4- Estructura de ficheros de la versión 1.6

En esta versión no se ha añadido ningún fichero más ni se a cambiado a grandes rasgos el funcionamiento de los mismos.

8.5-SSI-Dijsktra 1.6: Cambios realizados

En esta versión se realizaron bastantes cambios, ya que se tuvo que actualizar el SSI-Dijkstra a la nueva versión del UKB (1.6). Como ya se ha comentado anteriormente, se introdujeron nuevos tipos de palabras, por lo que se tuvieron que contemplar diferentes opciones para tratarlas de forma apropiada.

-Creación de getters nuevos para las palabras de tipo “cw_tgtword_nopv” (recordamos que son las palabras que deben ser desambiguadas pero que no tenían que ayudar en la desambiguación de las demás) y “ cw_ctxword” (recordamos que son las palabras que deben ser desambiguadas y ayudar a desambiguar al resto pero que no tienen que mostrarse en el resultado). Al introducir estos dos nuevos tipos, era necesario crear esos dos getters para saber qué tipo de palabras eran, ya que en función de cuál sería, había que realizar una u otra acción.

-Añadimos un nuevo vector, ya que el anterior no cubría nuestras necesidades.

-Modificación del constructor propio del UKB. Al introducir los nuevos tipos de palabras, hemos tenido que modificar el constructor que venía con el UKB para adaptarse a nuestras necesidades. Cuando recibía uno de los nuevos tipos de palabra, más concretamente el cw_tgtword_nopv, el nuevo constructor lo convertía a un tipo de palabra que a nosotros no nos interesaba, por lo que lo cambiamos para que siguiese manteniendo el tipo original.

-Creación de un nuevo método para mostrar los resultados (print_csent_simple_all). Con los nuevos tipos de palabras, creímos oportuno crear un nuevo método con el cual pudiésemos mostrar lo que nosotros quisiésemos. Para decidir qué se imprimía, decidimos crear una nueva opción que se tendría que especificar la hora de realizar la llamada.

-Modificación del algoritmo SSI-Dijkstra y SSI-Dijkstra+. Cuando una palabra del tipo “cw_tgtword_nopv” es desambiguada de forma normal o es monosémica, tiene que ser tratada de forma diferente, ya que éstas palabras solo nos interesa mostrarlas en el resultado. Éstas palabras no irían a la lista de las interpreted, como irían el resto.

-Creación de un algoritmo de ordenación. Se decidió crear dicho algoritmo por términos de eficiencia, ya que nos interesaba que las palabras de tipo “cw_tgtword_nopv” fuesen las últimas en desambiguarse.

-Creación de un método de control de errores. Con los nuevos tipos de palabras, se creó un caso particular en el que si todas las palabras fuesen del tipo “cw_tgtword_nopv”, dicho contexto sería imposible desambiguarlo por que se decidió avisar al usuario si ésto sucedía.

-Modificación de un try-catch en el main del SSI-Dijkstra . Cuando no se hacía la llamada de forma correcta, el programa terminaba de una manera brusca, por lo que se modificó para que terminase de la misma forma que el UKB.

9- Pruebas

A lo largo del proyecto, hemos ido haciendo diferentes pruebas que nos han ayudado a depurar los errores que han ido apareciendo. Las pruebas son un elemento fundamental para garantizar la calidad del software, y representa un revisión final de las especificaciones. Las pruebas descritas a continuación, sirven para mostrar la diferencia entre SSI-Dijkstra, SSI-Dijkstra+ y el UKB, con diferentes configuraciones y opciones.

Para calcular la precisión y el recall, comparamos nuestros resultados con un fichero de pruebas, el cual contiene las diferentes soluciones. Comparamos nuestras respuestas con dicho fichero para saber cuántas han sido correctas. Para saber cuales han sido correctas, utilizamos el siguiente método:

Tenemos el siguiente contexto a desambiguar:

ctx_00031921-n(belong_to#v#wf3#02719930-v)
00031921-n#n#wf0#2 00002137-n#n#wf2#2 belong_to#v#wf3#1 00356926-a#a#wf6#2 00001740-n#n#wf9#2 part#n#wf11#1 together#r#wf12#1

Observamos que *belong_to* ya tiene un sentido asignado (el *02719930)* pero aun así, también desambiguamos la palabra con el algoritmo elegido. Si el sentido que tenía asignado al principio coincide con el que a obtenido el algoritmo, significará que esa desambiguación ha sido correcta.

Las pruebas han sido realizadas con dos configuraciones diferentes: utilizando contextos desambiguados y sin usar contextos desambiguados. A continuación, pasaremos a explicarlas más a fondo:

Utilizando contextos desambiguados

Algunas palabras del contexto ya se proporcionan desambiguadas. Éstas son las que ayudarán a desambiguar el resto de palabras. Podemos observar en el siguiente ejemplo la composición de la misma:

ctx_00031921-n(belong_to#v#wf3#02719930-v)
00031921-n#n#wf0#2 00002137-n#n#wf2#2 belong_to#v#wf3#1 00356926-a#a#wf6#2 00001740-n#n#wf9#2 part#n#wf11#1 together#r#wf12#1

Sin utilizar contextos desambiguados

En el contexto sin desambiguar, todas las palabras del contexto se proporcionan sin desambiguar, salvo la glosa que tiene el sentido desambiguado (*00294868-n#n#wf0#2)*. Podemos observar en el siguiente ejemplo la composición de la misma:

ctx_00294868-n(hand#n#wf7#05564590-n)
00294868-n#n#wf0#2 slow#a#wf2#1 mode#n#wf3#1 locomotion#n#wf5#1 hand#n#wf7#1 knee#n#wf9#1 drag#v#wf11#1 body#n#wf13#1

9.1-Diferencias entre SSI-Dijkstra, SSI-Dijkstra+ y UKB

9.1.1-Utilizando contextos desambiguados

SSI-Dijkstra +

precision: 0.817 (762.00 correct of 933.00 attempted)
recall: 0.817 (762.00 correct of 933.00 in total)
attempted: 100.00 % (933.00 attempted of 933.00 in total)

P = 0.829 R = 0.817 F1 = 0.823
P = 762 / 919=0.829

R = 762 / 933=0.817

F1 = 2 * 0.829* 0.817/ (0.829+0.817)=0.823

SSI-Dijkstra

precision: 0.819 (764.00 correct of 933.00 attempted)

recall: 0.819 (764.00 correct of 933.00 in total)

attempted: 100.00 % (933.00 attempted of 933.00 in total)

UKB

P = 764/919=0.831

R = 764/933=0.818

F1 = 2*0.831*0.818/(0.831+0.818)=0.824

precision: 0.856 (799.00 correct of 933.00 attempted)

recall: 0.856 (799.00 correct of 933.00 in total)

attempted: 100.00 % (933.00 attempted of 933.00 in total)

P = 799/919= 0.869

R = 799/933=0.856

F1 = 2*0.869*0.856/(0.869+0.856)=0.862

Podemos observar que, en este caso, el UKB obtiene mayor precisión que ambos algoritmos, aunque a su vez, el SSI-Dijkstra tiene mayor precisión que el SSI-Dijkstra+. Se podría decir que son muy similares entre los tres, ya que obtienen unos resultados muy similares.

9.1.2-Sin utilizar contextos desambiguados

SSI-Dijkstra+

precision: 0.805 (751.00 correct of 933.00 attempted)
recall: 0.805 (751.00 correct of 933.00 in total)
attempted: 100.00 % (933.00 attempted of 933.00 in total)

P=751/919=0,817
R= 751/933=0,805
F1= 2*0,817*0,805/(0.817+0,805)=0,811

SSI-Dijkstra

precision: 0.797 (744.00 correct of 933.00 attempted)
recall: 0.797 (744.00 correct of 933.00 in total)
attempted: 100.00 % (933.00 attempted of 933.00 in total)

P=744/919=0,809
R=744/933=0,797
F1=2*0,809*0,797/(0,809+0,797)=0,803

UKB

precision: 0.854 (897.00 correct of 933.00 attempted)
recall: 0.854 (897 correct of 933.00 in total)
attempted: 100.00 % (933.00 attempted of 933.00 in total)

P = 897/919=0.867
R = 897/933=0.854
F1 = 2*0.867*0.854/(0.867+0.854)=0.860

Podemos apreciar que, al no utilizar contextos desambiguados, el UKB obtiene mejores resultados. La mayor diferencia la encontramos con el UKB y el SSID-Dijkstra, ya que este último pierde más

eficiencia. A su vez, el SSI-Dijkstra+ obtiene mejores resultados que el SSI-Dijkstra .

9.2-Opción POLY

Poly es una opción que nos permite ordenar las palabras por polisémia, por lo que analizaremos los resultados obtenidos en este caso con los anteriores.

9.2.1-Utilizando contextos desambiguados

SSID-Dijkstra+

precision: 0.824 (769.00 correct of 933.00 attempted)
recall: 0.824 (769.00 correct of 933.00 in total)
attempted: 100.00 % (933.00 attempted of 933.00 in total)

P = 769/919= 0.837
R = 769/933= 0.824
F1 = 2*0.837*0.824/(0.837+0.824)= 0.830

SSID- Dijkstra

precision: 0.819 (764.00 correct of 933.00 attempted)
recall: 0.819 (764.00 correct of 933.00 in total)
attempted: 100.00 % (933.00 attempted of 933.00 in total)

P = 764/919=0.831
R = 764/933=0.819
F1 = 2*0,831*0,818/(0,831+0,818)=0.824

Observamos que el SSID-Dijkstra+, con la opción poly, obtiene mejores resultados. Mientras que el SSID-Dijkstra obtiene los mismo, por lo que en este caso no sería necesario usar la opción para

ganar eficiencia.

9.2.2-Sin utilizar contextos desambiguados

SSID++

precision: 0.811 (757.00 correct of 933.00 attempted)
recall: 0.811 (757.00 correct of 933.00 in total)
attempted: 100.00 % (933.00 attempted of 933.00 in total)

P = 757/919= 0.824
R = 757/933= 0.811
F1 = 2*0.824*0.811/(0.824+0.811)= 0.817

SSI

precision: 0,798 (745.00 correct of 933.00 attempted)
recall: 0,798 (745.00 correct of 933.00 in total)
attempted: 100.00 % (933.00 attempted of 933.00 in total)

P = 745/919= 0.810
R = 745/933= 0,798
F1 = 2*0.810*0.798/(0.810+0.798)= 0.802

En este caso el SSID-Dijkstra+, con la opción poly, obtiene mejores resultados que sin ella. El SSID-Dijkstra obtiene una mejora insignificante, por lo que no se notaría si usamos la opción o no, hablando en términos de eficiencia.

10- Implantación

Una vez finalizado el proyecto en nuestra maquina, pasaremos a la fase de implantación en una nueva máquina.

Lo primero sería instalar el BoostGraph en dicha máquina, ya que sin esas librerías, es imposible ejecutar el SSID-Dijkstra. Tras eso, se compilaría el UKB, que tiene integrado el SSID-Dijkstra, y se comprueba que lo hace sin ningún problema. Una vez compilado, realizaríamos una serie de pruebas para comprobar que funciona correctamente. Si surgiese algún error, ya sea en la parte de compilación o en la de pruebas, buscaríamos la razón de dichos problemas y las soluciones para arreglarlo.

11- Gestión

Nos disponemos a resumir los aspectos más relevantes de la gestión del proyecto. En particular, veremos la diferencia entre las horas planificadas y las horas reales que hemos invertido a la hora de hacer el proyecto. Esto sucede porque la planificación inicial no suele coincidir con la real, ya que nunca se puede saber a ciencia cierta el tiempo que se tendrá que dedicar a cada cosa. Hemos optado por representar el tiempo en diferentes tablas y gráficas para poder apreciar de forma visual la diferencia entre ambas.

11.1-Comparativa entre esfuerzo planificado y real

11.1.1-Procesos tácticos

Procesos	*Planificado*	*Real*
Planificación	10	15
Gestión	5	4
Instalación	10	20
Reuniones	20	30
Total	45	69
Desviación	53%	

Tabla 3: Comparativa entre esfuerzo planificado y real en procesos tácticos

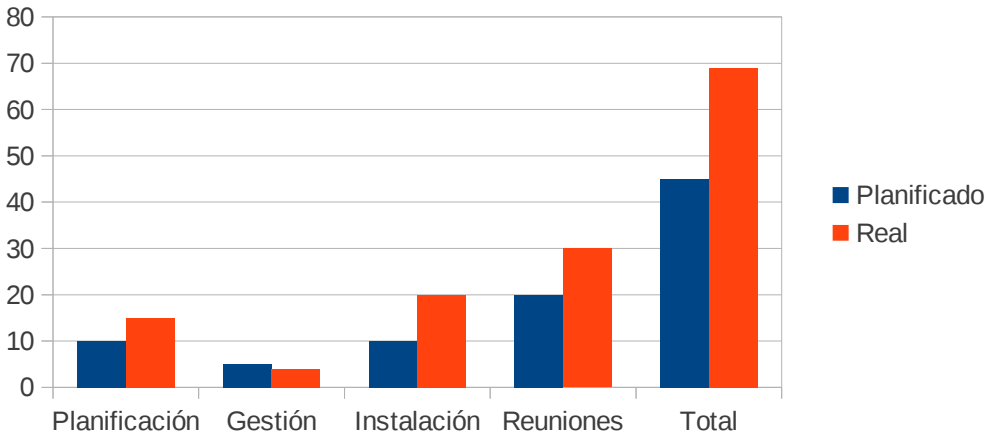


Figura 4: Esfuerzo planificado y real en procesos tácticos

Podemos apreciar que las tareas de instalación de herramientas y reuniones han sido las que más tiempo nos han llevado. El retraso en la instalación de herramientas, se debe a ciertos problemas que se tuvieron y que nos costó encontrar la solución. El problema surgió a la hora de instalar el BoostGraph, ya que, aun siguiendo los pasos necesarios para la instalación, no lo hacía de forma correcta. Al final se ha conseguido instalar gracias a la herramienta *Centro de software de Ubuntu*, ya que desde ahí no nos dió ningún problema. Respeto a las reuniones, tampoco se tenía una idea clara del número de reuniones que serían necesarias, por lo que se puso unas horas aproximadas que, por lo que podemos apreciar, no fueron suficientes.

11.1.2-Procesos operativos

1ºIteración

Procesos	Planificado	Real
Captura de Requisitos	6	7
Análisis	5	10
Diseño	7	3
Implementación	10	20
Pruebas	6	4
Total	34	44
Desviación	22,73%	

Tabla 4: Comparativa entre esfuerzo planificado y real en procesos operativos de la primera iteración

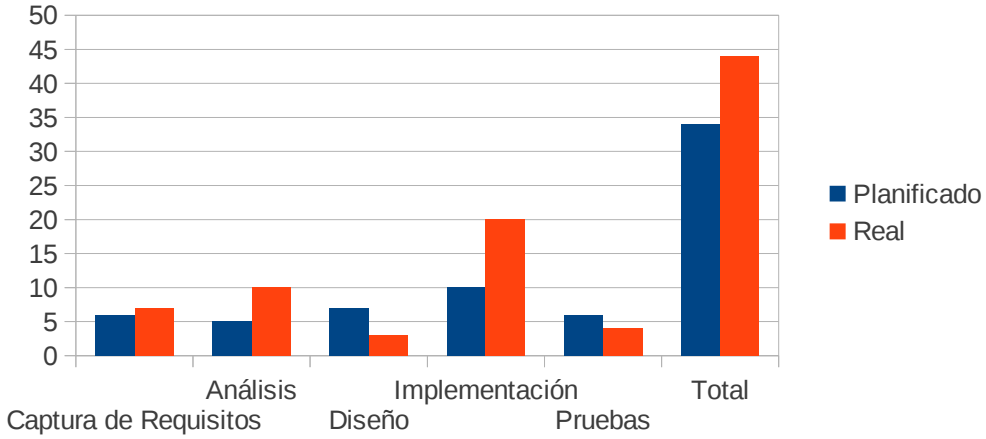


Figura 5: Esfuerzo planificado y real en procesos operativos de la primera iteración

Podemos observar que las tareas a las que se han dedicado más tiempo han sido el análisis y la implementación.

2ºIteración

Procesos	Planificado	Real
Captura de Requisitos	5	7
Análisis	5	6
Diseño	7	4
Implementación	10	8
Pruebas	5	6
Total	32	31
Desviación	3,22%	

Tabla 5: Comparativa entre esfuerzo planificado y real en procesos operativos de la segunda iteración

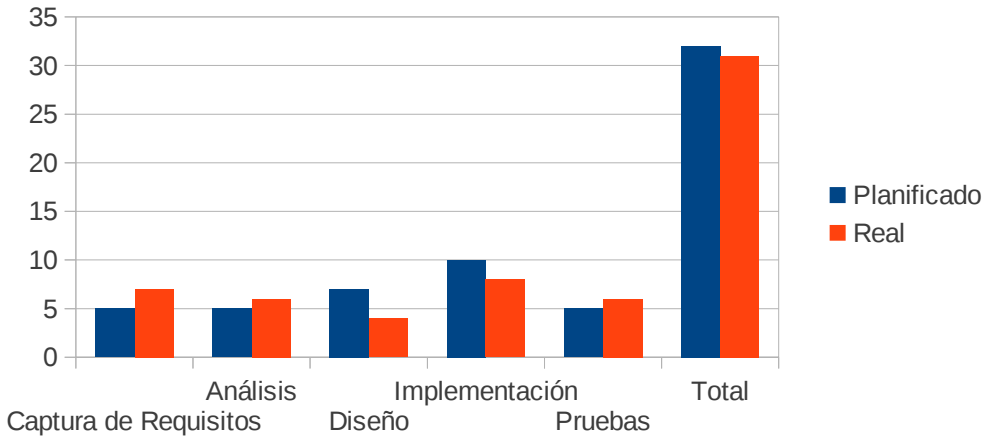


Figura 6: Esfuerzo planificado y real en procesos operativos de la segunda iteración

En esta iteración podemos observar que las horas planificadas y las reales son muy parecidas, en algún caso incluso menos. Ésto sucedió ya que, al actualizar la versión 1.5 a la 1.6, no se apreciaron cambios significativos que afectarán a nuestro proyecto.

3ºIteración

Procesos	Planificado	Real
Captura de Requisitos	5	6
Análisis	4	8
Diseño	5	4
Implementación	10	30
Pruebas	7	15
Total	31	63
Desviación	50,79%	

Tabla 6: Comparativa entre esfuerzo planificado y real en procesos operativos de la tercera iteración

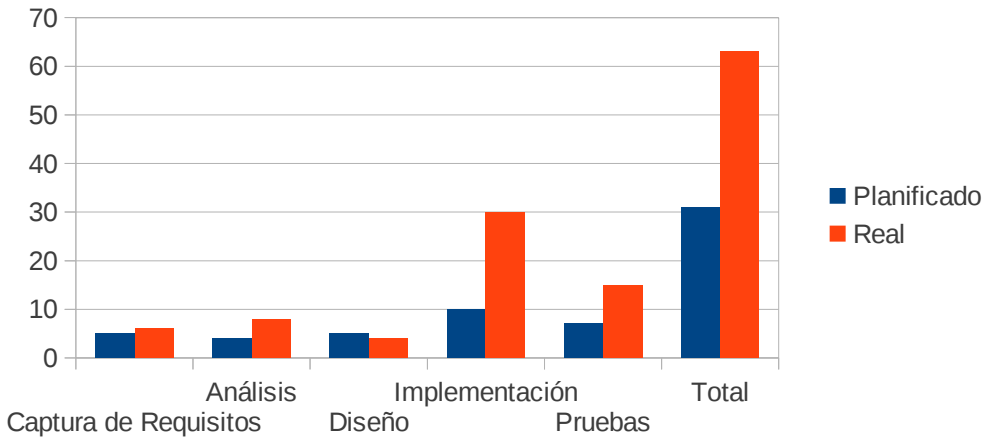


Figura 7: Esfuerzo planificado y real en procesos operativos de la tercera iteración

Podemos apreciar que las tareas más costosas de esta iteración han sido la implementación y las pruebas. Ésto sucedió, ya que se fueron introduciendo opciones y herramientas que, en un principio, no habían sido planteadas. Ésto también repercutió en las pruebas, ya que se tuvieron que hacer más de las necesarias.

11.1.3-Procesos Formativos

Procesos	Planificado	Real
C++	15	20
BoostGraph	15	10
UKB	20	30
Memoria	50	70
Presentación	7	
Total	107	130
Desviación	17,69%	

Tabla 7: Comparativa entre esfuerzo planificado y real en procesos formativos

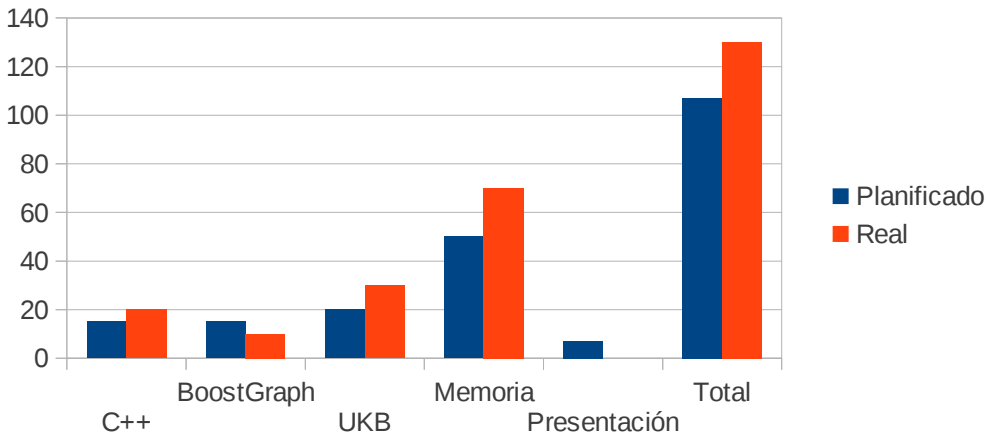


Figura 8: Esfuerzo planificado y real en procesos formativos

Se puede apreciar que la memoria y el UKB han sido las que más tiempo nos ha llevado. En general, se esperaba utilizar bastantes horas en la formación, ya que, como ya se comentó, el alumno desconocía la mayoría de las herramientas utilizadas.

11.2-Comparativa entre el esfuerzo planificado y real del proyecto

Procesos	Planificado	Real
Tácticos	45	69
Operativos	97	138
Formativos	107	130
Total	249	337
Desviación (%)	26,11%	

Tabla 8: Tiempo total de la comparativa entre esfuerzo planificado y real

Podemos observar que ha habido un incremento del 27,71% respecto al tiempo planificado. Ésto se debe a problemas inesperados y mala planificación.

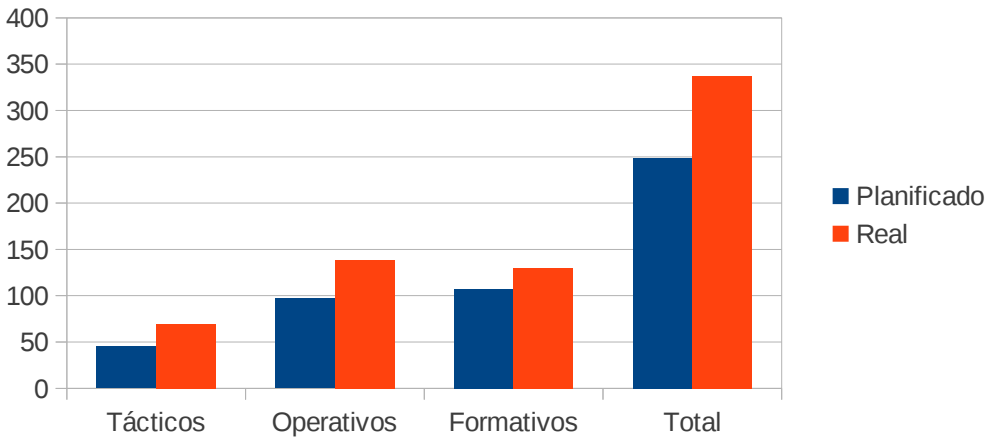


Figura 9: Esfuerzo planificado y real del proyecto

11.3-Justificación de las desviaciones

La mayoría de las desviaciones vienen de una no muy buena planificación, ya que muchas de las veces se asigna poco tiempo a tareas importantes. No se vió necesario hacer una replanificación, ya que se observó que se podía terminar en el plazo establecido.

Se puede observar cierta desviación a la hora de instalar las herramientas necesarias, ya que tuvimos bastantes problemas con ellas. Los primeros días de reuniones se dedicaron a intentar solucionar dichos errores. Ésto atrasó el proyecto, pero no vimos necesario replanificarlo, ya que se pudo terminar antes de que lo retrasara hasta un punto crítico. Todo ésto aumentó el tiempo planificado para las reuniones ya que fue un error inesperado y llevó tiempo solucionarlo.

Se dedicó bastante tiempo al aprendizaje del UKB, ya que el alumno no estaba muy familiarizado con el lenguaje, por lo que hubo partes que le costó entender, así como comprender el funcionamiento global del programa.

12- Conclusiones

12.1-Objetivos cumplidos

Podemos sentirnos satisfechos, ya que hemos cumplido todos los objetivos propuestos para este proyecto. Esperemos que éste algoritmo pueda ser añadido al UKB como una opción más para la desambiguación de palabras. En resumen, los objetivos cumplidos han sido los siguientes:

-Corrección de los algoritmos SSI-Dijkstra y SSI-Dijkstra +.
-Actualización del SSI-Dijkstra a la versión 1.6.
-Optimización y nuevas opciones.

Al principio, el proyecto me pareció algo complicado, ya que no es un área que se estudie o trate durante la carrera. Además, tenía que entender el UKB para poder realizarlo, pero una vez me centré en él y con la ayuda del director, las cosas se pusieron más claras. También tuvimos bastantes problemas a la hora de instalar el software necesario para el desarrollo, por lo que fue algo duro el comienzo. Desde el principio me pareció interesante, ya que había muchas cosas que nunca había usado y que quería aprender, puesto que durante la carrera no había tenido oportunidad.

Esta experiencia también me ha servido para aprender a gestionar un proyecto de éstas características de principio a fin. Como hemos podido ir viendo en el desarrollo, el tiempo planificado no se correspondía con el real, pero gracias a ésta experiencia, podemos establecer una precisión mayor en futuros proyectos.

En general, estoy bastante satisfecho con el proyecto, ya que se han cumplido todos los objetivos propuestos durante el mismo. También estoy contento de la oportunidad que he tenido de ampliar mi conocimiento hacia zonas que desconocía completamente.

12.2-Trabajos Futuros

En este apartado expondremos una serie de mejoras o trabajos a realizar en el proyecto.

12.2.1- Integración dentro del branch

Nos gustaría que UKB integrase el SSI para que en futuras versiones siga funcionando, ya que si el programa base se modifica o es actualizado, el SSI-Dijkstra puede dejar de funcionar y quedar obsoleto. Aún no sabemos si ésto será posible o no, pero es una opción bastante buena, ya que amplía el repertorio de opciones de UKB.

12.2.1- Actualización a la nueva versión del UKB

Al terminar el desarrollo del proyecto, nos enteramos de que había salido una nueva versión del UKB, más concretamente la versión 2.0. Estaría bien poder integrar el SSI-Dijkstra en esta nueva versión, para que pueda seguir siendo funcional y no quedar desactualizado.

13- Bibliografía

1. Todo lo relacionado con C++:
   a) http://www.cplusplus.com
   b) http://www.tutorialesenlared.com/
   c) http://c.conclase.net/
   d) http://msdn.microsoft.com/en-us/library/7

2. Boost
   a) http://www.boost.org/doc/libs/1_45_0/libs/graph/doc/table_of_contents.html

3. WordNet
   a) http://adimen.si.ehu.es/web/MCR
   b) http://adimen.si.ehu.es/cgi-bin/wei/public/wei.consult.perl

4. Definiciones e información generalmente
   a) http://www.wikipedia.org/

5. Ubuntu
   a) http://www.ubuntu.com/

6. Tutorial git
   a) http://schacon.github.com/git/gittutorial.html