Ismael García Varea

TRADUCCIÓN AUTOMÁTICA ESTADÍSTICA: MODELOS DE TRADUCCIÓN BASADOS EN MÁXIMA ENTROPÍA Y ALGORITMOS DE BÚSQUEDA

I.S.B.N. Ediciones de la UCLM
978-84-8427-504-6

Ediciones de la Universidad
de Castilla-La Mancha

Cuenca, 2007

Traducción Automática Estadística: Modelos de Traducción basados en Máxima Entropía y Algoritmos de Búsqueda

Tesis Doctoral

Presentada por Ismael GARCÍA VAREA
Dirigida por el Doctor D. Francisco CASACUBERTA NOLLA

Diciembre de 2003

Traducción Automática Estadística: Modelos de Traducción basados en Máxima Entropía y Algoritmos de Búsqueda

Trabajo realizado bajo la dirección del Doctor
D. Francisco CASACUBERTA NOLLA

Presentado para optar al grado de Doctor en Informática

Diciembre de 2003

Este trabajo ha sido subvencionado parcialmente por la Comisión Interministerial de Ciencia y Tecnología (C.I.C.Y.T., proyectos TAVAL, TIC-IFD1997-1433 y SISHITRA, TIC2000-1599-C02) y por la Comisión Europea (proyecto TransTypeII, IST-2001-32091).

Resumen

Este trabajo de tesis está dedicado en su totalidad a la traducción automática, una disciplina dentro de los campos de la inteligencia artificial y de la lingüística computacional y, más concretamente, dentro de la aproximación estadística a la misma. Todo el desarrollo de la tesis está dentro de la traducción basada en la regla de decisión de Bayes. Por ello se encuadra dentro del enfoque empirista de la traducción automática frente a uno más racionalista basado en técnicas lingüísticas.

En dicha aproximación podemos destacar tres ingredientes básicos en un sistema estadístico de traducción: un modelo de lenguaje, un modelo de traducción y un proceso de búsqueda, el cual podemos decir que intenta resolver el problema de la traducción propiamente dicho.

El primero no carece de interés, pero ya ha sido ampliamente estudiado con anterioridad, sobre todo por su aplicación a sistemas automáticos de reconocimiento del habla. Por ello en este trabajo nos limitaremos a hacer uso de las técnicas ya desarrolladas en modelado de lenguaje, concretamente en modelos de n-gramas dado su ya demostrado buen funcionamiento. De cualquier forma, dada su importancia en el tema y por completitud en la exposición, se dan una serie de conceptos elementales relacionados con estos modelos.

En lo que respecta a los modelos de traducción, en primer lugar, hemos de comentar que este trabajo está basado en lo que se conoce cómo modelos estadísticos de traducción basados en relaciones estructurales a nivel de palabra. En esta tesis hemos introducido modelos léxicos dependientes del contexto basándonos en la técnica de máxima entropía. Más concretamente hemos visto cómo desarrollar modelos de traducción basados en esta técnica, cómo integrarlos en los algoritmos de aprendizaje de modelos convencionales de traducción y cómo utilizarlos para mejorar las prestaciones de sistemas automáticos de traducción estadística.

En cuanto al problema de la búsqueda, hemos propuesto, diseñado y estudiado algoritmos de búsqueda siguiendo tres paradigmas clásicos para la resolución de problemas. Estos paradigmas son el de la programación dinámica, la ramificación y poda, y los algoritmos voraces. De todos ellos hemos hecho un estudio minucioso en lo que respecta a la eficiencia y calidad en la traducción, así como un estudio teórico y empírico de la complejidad de dichos algoritmos.

Todo este trabajo está apoyado por gran cantidad de experimentaciones reales, llevadas a cabo utilizando tareas clásicas y ampliamente utilizadas en el mundo de la traducción automática. Estas tareas son: la tarea del Turista, más conocida como la tarea EuTrans-I, aplicada a la traducción entre el castellano y el inglés; la tarea Hansards que se aplica a traducciones entre el francés y el inglés; y la tarea Verbmobil, tarea que aborda la traducción entre el alemán y el inglés.

Resum

Aquest treball de tesi està dedicat en la seua totalitat a la traducció automàtica, una disciplina dins dels camps de la inteligència artificial i la lingüística computacional i, més concretament, dins de l'aproximació estadística a la traducció. Tot el desenvolupament de la tesi està dins de la traducció basada en la regla de decisió de Bayes. Per això, s'emmarca dins de l'enfocament empirista de la traducció automàtica en front a un més racionalista basat en tècniques lingüístiques.

En l'arquitectura bayesiana podem destacar tres ingredients bàsics en un sistema estadístic de traducció: un model de llenguatge, un model de traducció i un procés de cerca, del qual en podem dir que intenta resoldre el problema de la traducció pròpiament dit.

No ens manca interés en el primer model, però ja ha estat àmpliament estudiat amb anterioritat, sobretot per la seua aplicació a sistemes automàtics de reconeixement de la parla. Per això, en aquest treball ens limitarem a fer ús de les tècniques ja desenvolupades en modelat de llenguatge, concretament en models de n-grames, vist el seu ja demostrat bon funcionament. Tanmateix, atesa la seua importància en el tema i per completitud en l'exposició, es donen una sèrie de conceptes elementals relacionats amb aquests models.

Pel que fa als models de traducció, en primer lloc, hem de comentar que aquest treball està basat en els que es coneixen com models estadístics de traducció basats en relacions estructurals a nivell de paraula. En aquesta tesi hem introduït models lèxics depenents del context basant-nos en la tècnica de màxima entropia. En particular, hem vist com desenvolupar models de traducció basats en aquesta tècnica, com integrar-los en els algoritmes de aprenentatge de models convencionals de traducció i com utilitzar-los per tal de millorar les prestacions de sistemes automàtics de traducció estadística.

Quant al problema de la cerca, hem proposat, dissenyat i estudiat algoritmes de cerca seguint tres paradigmes clàssics per a la resolució de problemes. Aquests paradigmes són el de la programació dinàmica, la ramificació i poda, i els algoritmes voraços. De tots ells n'hem fet un estudi minuciós pel que fa a l'eficiència i qualitat en la traducció, així com un estudi teòric i empíric de la complexitat dels algoritmes esmentats.

Tot aquest treball es recolza per una gran quantitat d'experimentacions reals, dutes a terme utilitzant tasques clàssiques i àmpliament utilitzades en el món de la traducció automàtica. Aquestes tasques són: la tasca del Turista, més coneguda com la tasca EuTrans-I, aplicada a la traducció entre el castellà i l'anglés; la tasca Hansards, que s'aplica a traduccions entre el francés i l'anglés; i la tasca Verbmobil, tasca que aborda la traducció entre l'alemany i l'anglés.

Abstract

This thesis is completely devoted to machine translation, a discipline which belongs to the fields of artificial inteligence and computational linguistics, and it is more specifically devoted to the statistical machine translation approach. The work presented here is based solely on the translation approach to Bayes' decision rule. Thus, this work can be classified as empiricist in contrast to a more rationalistic approach based on linguistic techniques.

We can distinguish three main different ingredients within this approach to machine translation: a language model, a translation model, and a search decoding problem. The latter can be considered the sheer problem of machine translation.

The first ingredient, the language model, is not treated here in full, mainly because it has been widely studied and used in the field of automatic speech recognition. Due to this, we only make use of the techniques already developed in language modelling and, more specifically, we use the well-known n-gram models. Nonetheless, because their relevance to the problem addressed in this thesis, we do introduce the basic and necessary concepts of language models.

With respect to translation models, it must be said first that this work is based on single-word statistical translation models. We have introduced context-dependent lexicon models based on maximum entropy techniques. In particular, we have seen how to develop maximum entropy translation models, how to integrate them into the training algorithms of conventional translation models, and how to use them in order to improve the performance of statistical machine translation systems.

Regarding the search decoding problem, we have proposed, designed, and studied several search algorithms following three classical problem solving paradigms. Those paradigms are: dynamic programming, the branch and bound approach, and the greedy algorithms approach. For all of them we have performed a detailed study concerning efficiency and translation quality. Additionally, a study of computational and empirical complexity has been done.

All this work has been supported by a large quantity of experiments, which were carried out on three well-known tasks in the field of machine translation. These tasks are: the tourist Spanish-English task, better known as EuTrans-I task, the French-English Hansards task, and the German-English Verbmobil task.

Prólogo

Una de las capacidades más fascinantes que posee el hombre, desde prácticamente sus orígenes, y que definitivamente lo hace distinto y superior a cualquier otro ser vivo, es la capacidad de hablar y de intercambiar información entre distintos individuos de su misma especie.

Sin duda alguna, esta capacidad de comunicación ha hecho que durante nuestra existencia en la Tierra, los conocimientos y pensamientos de millones y millones de seres humanos hayan perdurado en el tiempo y se hayan transmitido generación tras generación, para poder obtener grandes avances científicos, tecnológicos y culturales hasta nuestros días.

Por otra parte, esta transferencia de información, sobre todo plasmada en documentos escritos, se ha visto mermada en gran medida por las diferencias lingüísticas de las distintas razas y culturas existentes en el mundo. Sin ir más lejos aún existen serias dudas y distintos criterios de cómo descifrar los jeroglíficos egipcios y otros documentos escritos en lenguas muertas hace siglos.

En definitiva, hemos llegado a un punto en el que la transferencia de información es el eje sobre el que se mueve el mundo, y de hecho, ya prácticamente todo el mundo asume que estamos viviendo en la famosa sociedad de la información.

A pesar de que hoy en día las barreras lingüísticas se van limando poco a poco, principalmente por la disponibilidad de los medios existentes para el estudio y aprendizaje de otras lenguas, es de destacar la gran necesidad que existe de traducir textos entre distintos idiomas, sobre todo en sociedades plurilingües como en las que prácticamente todo el mundo se desarrolla hoy en día. Esta necesidad de traducir textos se hace cada vez más importante debido a la ingente cantidad de documentos que a diario se desarrollan en distintas lenguas, y que se ponen de manifiesto por infinidad de medios audiovisuales.

Todos estos factores nos han influenciado y conducido a dedicar nuestra investigación en esta línea. Por tanto, la motivación para la realización de este trabajo, aparte de nuestros gustos personales acerca del tema, podría resumirse teniendo en cuenta los siguientes aspectos:

- El mundo cada día más plurilingüe en el que vivimos y nos desarrollamos, no solamente dentro de ambientes culturalmente privilegiados sino a cualquier nivel social.

- La necesidad existente de obtener y manejar textos en otros idiomas, principalmente motivada por la rápida e incontrolada expansión que el mundo de Internet ha sufrido en los últimos años.
- Facilitar la interacción hombre-máquina que el uso cotidiano del ordenador personal nos ha obligado.
- La accesibilidad a personas discapacitadas al uso de ciertas máquinas y herramientas hasta hoy en día impracticables.

Todos estos motivos unidos a la posibilidad de que disponemos de automatizarlos haciendo uso de potencia de las computadoras actuales, creemos que queda suficientemente justificado el porqué de este trabajo y de cualquier otro encaminado en definitiva a mejorar y facilitar la calidad de vida del hombre.

En esta de tesis hemos pretendido no solo mostrar el trabajo de investigación desarrollado durante varios años en el ámbito de la traducción automática, sino también el proporcionar un texto de referencia en el que se ha intentado plasmar algunas ideas generales acerca de la traducción automática, profundizando en la aproximación estadística a la misma. Por esta razón, algunas partes de este trabajo son recopilaciones de otros trabajos que, a nuestro criterio, han sabido mostrar de forma clara, precisa y, lo más importante, intuitiva, acerca de algunos conceptos tradicionalmente bastante complejos para lectores que no estén muy familiarizados con el tema. Concretamente, parte del Capítulo 2 (Preliminares. Conceptos básicos) en el que se presenta de una forma altamente intuitiva el problema de estimación y aprendizaje de modelos estadísticos de lenguaje y de traducción, ha sido recopilado de un texto titulado: "A statistical machine translation tutorial" escrito por Kevin Knight en 1999. Del mismo modo, la introducción a los modelos por máxima entropía (Sección 4.2) ha sido extraída, prácticamente en su totalidad, de un articulo magistral acerca de esta aproximación titulado "A maximum entropy approach to natural language processing" de Alan Berger y colaboradores, publicado en "Computational Linguistics" en 1996. Evidentemente, podríamos haber obviado estas partes dentro de la memoria de esta tesis, pero hemos considerado oportuno incluirlas precisamente por el motivo expresado al principio de este párrafo, proporcionar al lector un texto de referencia intentando expresar las ideas principales de la forma más intuitiva posible. Esperamos que este esfuerzo sirva de provecho sobre todo a aquellos investigadores que se embarquen por primera vez en el mundo de la aproximación estadística a la traducción automática, con ese propósito lo hemos hecho.

Esta memoria se ha dividido en cinco partes y ocho capítulos que pasamos a describir a continuación:

- Parte I: Introducción y conceptos básicos. Esta parte contiene dos capítulos:
  - Capítulo 1. "Introducción". En este capítulo se ha expuesto una introducción general a la traducción automática, exponiendo una breve

reseña histórica, así como una clasificación actualizada de los sistemas automáticos de traducción. También se hace una primera introducción a la aproximación estadística a la traducción automática, exponiendo el estado del arte de dicha aproximación. Por último se describen los objetivos que se han pretendido cubrir con este trabajo.

- Capítulo 2. "Traducción automática estadística. Preliminares". Este capítulo lo hemos dedicado para introducir una serie de conceptos básicos que caracterizan a la traducción automática estadística frente a otras aproximaciones. Asimismo, se dan unas ideas generales acerca de modelos de traducción y de lenguaje, así como la descripción de los corpus utilizados en el desarrollo de esta memoria y la forma de evaluar la traducción. Este capítulo podrá ser pasado por alto por aquellos lectores familiarizados con el tema. Quizás, la única parte de obligada lectura de este capítulo sería una sección que hemos titulado "Errores en el proceso de búsqueda. Modelo de error", por ser un concepto necesario para entender algunos comentarios expuestos a lo largo del resto de capítulos.

- Parte II: Modelos de traducción. Esta parte también contiene dos capítulos:

  - Capítulo 3. "Modelos estadísticos de alineamiento". En este capítulo se exponen con cierto nivel de detalle los modelos estadísticos de alineamiento que se utilizaran en el resto de la tesis.
  - Capítulo 4. "Modelos de traducción basados en máxima entropía". En este capítulo es donde realmente empiezan las aportaciones importantes de esta tesis. En él se expone la aproximación por máxima entropía a la traducción automática, la cual será utilizada para definir modelos léxicos dependientes del contexto, entrenarlos, integrarlos en el entrenamiento de los modelos clásicos de alineamiento de IBM, y utilizarlos para mejorar sistemas estadísticos de traducción automática en general.

- Parte III: Algoritmos de búsqueda. En esta parte se exponen todos los algoritmos de búsqueda desarrollados en esta tesis. Esta parte contiene tres capítulos, uno por cada aproximación utilizada para abordar el problema de la búsqueda en la traducción automática estadística. En cada uno de ellos se exponen las características básicos de los algoritmos y se hace un estudio teórico de la complejidad computacional de los mismos. También se realiza un estudio empírico de eficiencia y calidad en la traducción. Concretamente esta parte contiene los siguiente capítulos:

  - Capítulo 5: "Algoritmos de búsqueda basados en programación dinámica". Como su título indica, en él se desarrollan una serie de algoritmos basados en esta técnica.

  - Capítulo 6: "Algoritmos de pila". En este capítulo se expone la familia de algoritmos basados en pila más comúnmente utilizados. Se hace hincapié en el estudio de la eficiencia y calidad, en la comparación entre ellos.
  - Capítulo 7: "Algoritmos de búsqueda voraces" Al igual que los dos anteriores, en este capítulo se expone una familia de algoritmos que utilizan técnicas de búsqueda locales.

- Parte IV: Conclusiones y bibliografía. Esta parte contiene el Capítulo 8: "Conclusiones", en el cual se resumen las aportaciones más relevantes de esta tesis, así como las principales conclusiones que podemos sacar de este trabajo. A continuación podemos encontrar la relación de citas bibliográficas referenciadas en el toda la exposición de esta memoria.

- Parte V: Apéndices. En el apéndice A exponemos una breve descripción de la traducción de voz desde el punto de vista estadístico. En el apéndice B se describen criterios de búsqueda para algoritmos basados en programación dinámica no desarrollados en el grueso de la tesis. El apéndice C, y último, se expone la experimentación llevada a cabo para el establecimiento de los parámetros de optimización, para los algoritmos de búsqueda, para la tarea Hansards.

Por último debemos comentar que este trabajo no debe de considerarse como un trabajo cerrado sino todo lo contrario: consideramos que es un punto de partida de una línea de investigación (a nuestro juicio apasionante) de la que aún queda mucha tela que cortar. Con esta idea invitamos al lector a leer esta tesis esperando que suscite en él el mismo grado de interés que otros trabajos en esta línea nos suscitaron a nosotros y que de alguna forma han hecho posible éste. No nos cabe duda que el trabajo aquí expuesto sea susceptible de mejoras, así que invitamos al lector a proponerlas, estaremos encantados de oír sus comentarios. Del mismo modo también esperamos sepa perdonar los fallos y/o erratas que encuentre en su lectura.

Agradecimientos

Como siempre que he tenido que escribir este apartado en trabajos previos a éste, me viene a la mente gran cantidad de gente sin la cual este trabajo no podría haberse llevado a cabo. Evidentemente no se trata aquí de mencionarlos a todos pues ellos de sobra lo saben.

No obstante, en primer lugar quiero expresar mi más sincero agradecimiento al director de esta tesis, Paco Casacuberta, por la gran cantidad de ideas que me ha ofrecido, planteándolas como un reto, pero apoyándome continuamente. También por haberme introducido en el mundo de la investigación, haberme enseñado gran parte de lo que sé, por haber confiado en mí y haberme ofrecido la posibilidad de trabajar conjuntamente en varios proyectos de investigación relacionados con el tema. Por todo esto último, hago extensivo mi agradecimiento a Enrique Vidal, codirector del grupo de investigación.

Por otra parte tengo que agradecer la ayuda prestada por Hermann Ney en todo lo referente a aspectos técnicos de este trabajo por las innumerables charlas que hemos mantenido en los períodos en que estado trabajando en su departamento en Aachen (Alemania), por los consejos ofrecidos, por el desinteresado ofrecimiento de material y herramientas de trabajo que siempre han estado a mi disposición y por ofrecerme la posibilidad de colaborar con su equipo de trabajo. Dentro de este equipo de trabajo tengo que hacer mención especial a Franz J. Och, sin el que definitivamente este trabajo no se hubiese llevado a cabo, por la ayuda inestimable que siempre me ha ofrecido, no solo a nivel profesional sino a nivel personal en algún momento crítico.

Otra persona a destacar ha sido Dani Ortíz, primero alumno y ahora amigo, por su inestimable trabajo en el desarrollo e implementación de los algoritmos de pila y por su desinteresada ayuda en el desarrollo de la fase experimental de este trabajo. Gracias por haberme dejado abusar de nuestra confianza.

Tampoco puedo olvidar a mis compañeros del PRHLT-DSIC y a los del departamento de informática de la UCLM, por su apoyo incondicional en los momentos de flaqueza y por último a todos los amigos (especialmente a Juan Antonio Guerrero y José Hernández Orallo) que me han ayudado a mejorar la redacción de esta tesis sin los que sin duda alguna pasaría a acumular polvo en alguna estantería del DSIC.

Ya fuera del ámbito académico, y no por eso menos importantes para mi,

también quiero dar las gracias a mi familia, sobre todo a mis padres por haberme dado el ser y casi todo lo que soy, y que junto a mi hermana, han hecho de mi vida una ilusión. Junto a ellos también tengo que agradecer a mis padres y hermanos políticos el haberme apoyado en todo momento, sobre todo durante estos últimos meses tan duros para todos.

Desgraciadamente este trabajo ha visto la luz demasiado tarde, pues la persona con la que más me hubiese gustado compartir tan emotivo momento, ya no se encuentra entre nosotros. Quiero dejar claro que su ayuda ha sido determinante en la consecución de este trabajo. Esa persona ha sido Charo, mi amiga, compañera, esposa, amante y también madre de mi hijo. Gracias por haberme dado tu calor, apoyo, cariño y amor durante todos estos años. Sinceramente, sin ti no hubiese podido ser.

Por último, también tengo que agradecer la ayuda que, desde muy recientemente, me está prestando mi hijo Ismael, que aunque él todavía no es consciente de ello, hace que permanezca en mi la ilusión de seguir viviendo proporcionándome día a día ánimos para poder seguir adelante.

Sinceramente, ¡gracias a todos!

Índice General

Índice de Figuras

Índice de Tablas

Índice de algoritmos

I

Introducción y conceptos básicos

Capítulo 1

Introducción

El tema principal de esta tesis es la traducción automática, y dentro de ésta, concretamente lo que se conoce como *aproximación estadística a la traducción automática*. En este capítulo de introducción, aparte de destacar el porqué de la necesidad de la traducción automática y de introducir el concepto de traducción, haremos un pequeño bagaje por la historia de la traducción automática en general, comentaremos las distintas aproximaciones utilizadas y por último nos centraremos en el tema principal de esta tesis, la traducción automática estadística. En ésta última parte haremos una pequeña introducción a la traducción automática estadística y comentaremos con cierto nivel de detalle el estado del arte de dicha aproximación. Por último expondremos los objetivos que se han pretendido cubrir en este trabajo de tesis.

1.1 Importancia de la traducción automática

En [Abeitua 01], podemos encontrar una buena referencia con bastante nivel de detalle del porqué es necesaria la traducción automática teniendo en cuenta el entorno multicultural en el que nos desarrollamos. Veamos algunas ideas expuestas en dicho trabajo.

La sociedad actual, también denominada Sociedad de la Información, es y seguirá siendo plurilingüe, por lo que la traducción de los datos se erige como el principal cuello de botella para la pretendida globalización de la información. Comparado con las rotativas más modernas capaces de producir hasta unos 20 millones de páginas por hora, un traductor manual puede llegar a transcribir, en los casos más favorables, unas 20 páginas por día, mejorando muy poco la

productividad del monje copista medieval que transcribía 3 o 4 páginas en ese mismo periodo de tiempo [Abeitua 01].

Sin ir más lejos, el problema ya quedó patente con la incorporación de nuevos estados miembros a la Unión Europea. Por ejemplo, cuando Suecia y Finlandia se incorporaron a la UE en 1995, hubo que traducir alrededor de 60.000 páginas de regulaciones comunitarias, labor que se prolongó excesivamente debido a la poca efectividad de los traductores disponibles.

Otro ejemplo lo encontramos en [Murray Jr. 66], donde se revela que se registran aproximadamente 20 millones de palabras por día de información técnica. Un lector capaz de leer 1.000 palabras por minuto necesitaría 45 días, a una media de 8 horas diarias, para digerir la producción de un día. Al cabo de estos 45 días, su desfase sería de 5,5 años. Asimismo, una comunidad lingüística necesitaría de 2.000 esforzados traductores para poder asimilar sin retardo este caudal diario de información técnica.

La mayor parte de los traductores profesionales producen texto que no tiene un alto valor cultural ni literario, ya que simplemente se dedican a satisfacer la enorme y creciente demanda de traducciones de documentos muy diversos: estudios técnicos y científicos, transacciones comerciales, informes administrativos, documentación jurídica, manuales de instrucciones, libros de texto de medicina o agricultura, patentes industriales, panfletos publicitarios, reportajes periodísticos, etc. Una parte de este trabajo resulta difícil y constituye un reto, mientras que un alto porcentaje es tedioso y repetitivo.

Por otro lado, la demanda de estas traducciones se está incrementando a un ritmo superior a como lo hace la capacidad de los traductores, por lo que la ayuda del ordenador en el proceso de traducción se vuelve casi indispensable.

1.2 El concepto de traducción

La primera idea que nos viene en mente al hablar de traducción es la de un mecanismo que permita la comunicación oral o escrita entre dos personas cuando ambas no conocen la lengua o el idioma de la otra. De hecho la definición de la palabra traducir, según el diccionario de la Real Academia Española dice:

> **Traducir.**(Del lat. *traducere*, hacer pasar de un lugar a otro). tr. Expresar en una lengua lo que está escrito o se ha expresado antes en otra. || 2. Convertir, mudar, trocar. || 3. Explicar, interpretar. MORF. conjug. c. *conducir*.

Parándonos a pensar un poco, muchas tareas más o menos comunes en nuestras vidas no dejan de ser meras traducciones dependiendo del contexto en que lo apliquemos, es decir dependiendo de lo que entendamos por lengua/lenguaje fuente y destino. Por ejemplo, la acción de andar se puede entender como un proceso de traducción en el sentido de que los impulsos nerviosos que emite el cerebro se traducen en movimiento cuando entran en acción los músculos de las

extremidades. Lo mismo ocurre con gran cantidad de fenómenos naturales como las reacciones químicas que continuamente el cuerpo humano realiza, la generación/transformación de luz eléctrica, los fenómenos atmosféricos, etc. Ya desde un punto de vista más computacional, el proceso de compilación o interpretación de un lenguaje de programación no es más que un proceso de traducción de un lenguaje de alto nivel a otro de bajo nivel, como lo es el lenguaje máquina.

De las distintas formas que tenemos de ver una traducción, en este trabajo nos vamos a centrar en la traducción propiamente dicha, es decir intentaremos resolver el problema de traducir textos de un idioma a otro, aunque por contra de lo que podemos pensar a priori lo intentaremos resolver de forma automática, es decir utilizando una máquina.

La traducción desde este punto de vista, y siguiendo la aproximación utilizada en esta tesis, está íntimamente ligada a los conceptos de reconocimiento automático del habla (RAH) y reconocimiento de formas. Dado que estos dos conceptos están bastante estudiados, nos vamos a aprovechar de algunas de sus características para desarrollar nuestro sistema de traducción. Evidentemente no vamos a desarrollar aquí estos dos conceptos, ya que en principio se sale de nuestros objetivos, y además existe una amplia bibliografía acerca del tema, por ejemplo [Casacuberta and Vidal 88, Jurafsky and Martin 00, Duda et al. 00, Huang et al. 01, Manning and Schütze 01, Quatieri 02].

De cualquier modo un sistema de RAH lo podemos ver como una máquina capaz de aceptar señales acústicas, procesarlas y producir una serie de acciones que el sistema pueda realizar. Entonces desde este punto de vista, la comprensión del lenguaje la podemos ver como una traducción entre dos lenguajes. Uno, el de entrada, que será el lenguaje natural y otro sería el propio lenguaje de esa máquina.

Por tanto, y dejando ya aparte el problema de captación de voz, nos basaremos en las técnicas ya estudiadas y probadas de los sistemas de RAH, y dentro de ellas las conocidas como técnicas estadísticas, para desarrollar un sistema de traducción automática.

1.3 Un poco de historia

Las primeras ideas de automatizar la traducción de lenguas aparecieron en el siglo XVII cuando algunos filósofos, matemáticos y lingüistas pensaron en la idea de un diccionario automático que suavizara/rompiera de alguna forma las diferencias/barreras lingüísticas de la época. Evidentemente, estas ideas no se pudieron poner en práctica hasta la aparición de los primeros computadores allá por 1940.

El concepto de traducción de un lenguaje a otro mediante una máquina es más antigua incluso que los propios ordenadores. Según Yehosua Bar-Hillel [Ban and Feigenbann 81], uno de los primeros investigadores en este campo, la idea de traducción automática fue concebida a principios de los años 30 por el

soviético P.P. Smirnov-Tropansky y por el francés G.B. Artooni. El trabajo de éstos nunca recibió mucha atención, quedando latente hasta una década después cuando el ambiente era mucho más favorable debido a la aparición de los primeros computadores digitales. En algunos centros científicos del mundo, la gente creyó (con cierta justificación) que los computadores podrían conducir a ideas enteramente nuevas e insospechables acerca del hombre y (quizás perdiendo justificabilidad) que los computadores podrían ayudar a producir un nuevo mundo. En pocas palabras, había grandes expectativas acerca del potencial de estas "Máquinas Pensantes", y rápidamente fueron aumentando aún más. Sobre todo, cuando durante la Segunda Guerra Mundial se utilizaron en tareas de estrategia como descifrar códigos y calcular complicados cálculos matemático-nucleares, en las que se obtuvieron resultados verdaderamente convincentes. Esto ocurrió a la vez que Claude Shannon formuló sus ideas acerca de la Teoría de la Información, a la vez que Norbert Wieman inventó el concepto de Cibernética, y cuando Pitts y McCullough desarrollaron sus primeras ideas acerca de redes neuronales y funciones cerebrales.

Debemos pensar que cuando se empezó a trabajar en traducción automática, la programación se hacía mediante tarjetas perforadas y el único lenguaje de programación era el lenguaje máquina. Conceptos como "array" y subrutina no habían aparecido, ni mucho menos conceptos como pilas, colas, compiladores de lenguajes, procedimientos recursivos, etc. Además, aún no se conocían las gramáticas contextuales e incontextuales. Cuando los computadores se empezaron a utilizar en el estudio del lenguaje, se hacia mediante experimentos estadísticos como por ejemplo recopilación de matrices de frecuencia de letras y de frecuencia de transición entre letras. Cada matriz podía ser usada para producir interesantes muestras de Pseudo-Lenguajes produciendo palabras, de letras generadas de forma aleatoria, con las mismas características que palabras inglesas.

Los primeros estudios de una forma seria de lo que nosotros entendemos por traducción automática, datan de una serie de discusiones entre Warren Weaver y A. Donald Booth en 1946 [Ban and Feigenbann 81]. Estos hombres estaban familiarizados con trabajos de decodificación por ordenador, basados en tablas de frecuencias de aparición de letras y palabras. Creyeron que estos mismos métodos podían ser aplicados a la traducción y que el principal obstáculo sería incorporar un diccionario completo de los dos lenguajes. Por supuesto, sabían que simplemente con tener un diccionario no resolverían el problema, ya que había que tener en cuenta otros problemas como:

- Que muchas palabras tienen varias traducciones dependiendo del contexto.
- Que el orden de las palabras en una frase varía de un lenguaje a otro.
- Y que las frases hechas no podían ser traducidas palabra a palabra sino como un todo.

No obstante parecía admisible, en aquellos días, que el mayor de los problemas de la traducción fuese el del vocabulario y por tanto al menos una gran parte de la traducción parecía mecanizable.

En 1947 Booth y D.M.V. Britten obtuvieron un programa para consultar un diccionario detallado, en el que cada variante de una palabra tenía su entrada en el diccionario y en 1948 R.M. Richens sugirió la adición de reglas acerca de las inflexiones de las palabras, de modo que la redundancia de las entradas del diccionario podía ser eliminado.

La primera aproximación a la TA apareció a finales de los años 40 en un memorándum titulado *"Translation"* presentado por Warren Weaver [Weaver 55] en 1949. En este artículo, investigadores de IBM presentaron la idea de la TA al público proponiendo varios métodos de cómo atacar el problema, variando desde análisis estadísticos hasta la exploración de la lógica y ciertas propiedades universales de los lenguajes. Además de la idea de que todos los lenguajes tienen muchas características en común, otras tres ideas contenidas en este documento tienen un valor relevante. Éstas son:

1. La primera es la noción de una "ventana" a través de la cual se pueden ver exactamente 2N+1 palabras del texto; Weaver sugirió que si N es lo bastante grande, se puede determinar la traducción correcta de la palabra que está justo en el medio de la ventana, N debería ser una función de palabra, en vez de una constante, y que el valor de N debería ser tal que el 95% de las palabras pudieran ser traducidas correctamente el 98% de las veces.

2. La segunda idea importante viene a ser la siguiente frase: "Cuando veo un artículo en ruso digo: Esto realmente está escrito en inglés, pero ha sido codificado mediante unos símbolos extraños. Procederé a decodificarlo". Esto quiere decir que el concepto de Texto Origen y Texto Traducido dicen la misma cosa.

3. La tercera idea parte de la anterior y dice que la traducción entre los lenguajes A y B significa ir de A a un lenguaje intermedio "Lenguaje Universal" (*Interlingua*) que supuestamente todos los humanos comparten, y después a B. Esta idea de una representación intermedia de la semántica (o significado de una palabra) aparece a menudo en los trabajos modernos de procesamiento del lenguaje natural en Inteligencia Artificial, bajo la rúbrica de Representación del Conocimiento.

Después del Memorándum de Weaver, se empezó a trabajar en varios centros de los Estados Unidos. Erwin Reiffer concibió la idea de dos funciones auxiliares realizadas por seres humanos, llamadas Pre-Editor y Post-Editor. El Pre-Editor prepararía la entrada del texto dejándolo tan libre como fuera posible de ambigüedades y otras dificultades, y el Post-Editor tomaría el texto traducido por la máquina y lo convertiría a texto gramaticalmente comprensible.

De una conferencia en 1952 se extrajo suficiente información como para implementar un programa eficiente para consultar un diccionario y trabajar hacia la invención o descubrimiento del hipotético Lenguaje Universal, llamado "Machinese", el cual fue propuesto por Weaver como el lenguaje intermedio para los traductores automáticos.

A.G.Oettinger fue uno de los pioneros en diseñar un programa que sacara traducciones de palabra a palabra del ruso al inglés. Gran cantidad de palabras rusas tienen más de una posible traducción, así todas ellas eran listadas en el texto inglés de salida, encerradas entre paréntesis. Claramente esto aún estaba lejos de la idea de traducción automática, pero ya se empezaban a obtener los primeros resultados.

En los dos años siguientes, la mayoría de los esfuerzos estaban orientados hacia inventar el modo de manejar los diferentes finales de las inflexiones de las palabras y en estimar el tamaño del vocabulario necesario para traducciones de diferentes grados de calidad.

En pocos años fueron muchos los proyectos de TA que se empezaron en los Estados Unidos, hasta que en 1954 (mismo año en que se fundó la revista "Machine Translation"), la universidad de Georgtown mostró por primera vez en público el primer sistema de traducción automática. Dicho sistema no tuvo mucho impacto científico debido a que era un sistema aplicado a un dominio muy restringido (49 frases del ruso al inglés) que consistía en un vocabulario de unas 250 palabras. A pesar del poco éxito alcanzado hasta el momento, fue en esa época cuando la guerra fría estaba en pleno apogeo y había un gran interés en la posibilidad de poder realizar traducciones automáticas de forma rápida y precisa. Esto llevó al gobierno de EEUU a invertir gran cantidad de dinero en investigación en este campo.

Esta inversión económica se mantuvo durante toda la década de los 50. Durante esta década las aproximaciones a la TA siguieron dos vertientes completamente distintas, que por aquel entonces se les conocía como aproximación por fuerza bruta y aproximación perfeccionista. Los primeros abogaron por soluciones rápidas basadas en métodos empíricos de ensayo y error, mientras que los segundos abogaron por soluciones más a largo plazo basadas en métodos lingüísticos. De cualquier forma la investigación llevada a cabo por estas dos vertientes durante esta década no llegó a buen puerto y los resultados fueron mucho peores de los esperados, aunque en realidad esto mismo ocurrió en el campo de lingüística computacional e incluso en el campo de la inteligencia artificial.

En los años 60 empezó la época de la decadencia en el campo de TA. A pesar de que las predicciones habían augurado un inminente avance en el campo, la desilusión creció conforme la complejidad que los problemas lingüísticos involucrados se hacían aparentes. Para estas fechas algo estaba claro y era que la idea de crear un sistema de TA capaz de obtener buenas traducciones no era tan simple como pudo parecer en un principio. Al final, el gran entusiasmo aparecido en la década de los 50 se vio truncado por la aparición de dos informes publicados

en los 60: el informe de Bar-Hillel en 1960 [Bar-Hillel 60] y el informe ALPAC en 1966 [Pierce and others 93]. En su informe, el investigador Bar-Hillel concluyó que crear un sistema completamente automático de traducción de alta calidad (FAHQT[1]) era un objetivo demasiado ambicioso, y que la única forma de superar las dificultades de la TA conllevaría incluir gran cantidad de información acerca del mundo. Por lo tanto Bar-Hillel recomendó que la investigación en el campo de la TA debería concentrarse en objetivos menos ambiciosos que la FAHQT, como por ejemplo sistemas automáticos de ayuda a la traducción. Por otra parte el informe ALPAC fue mucho más drástico. Este informe concluyó que la TA era lenta, de muy baja calidad y el doble de cara que la traducción humana. A pesar de que este informe fue duramente criticado por ser intolerante, sesgado y poco serio, el efecto que causó provocó la virtual desaparición de la investigación en TA en los Estados Unidos durante más de una década.

A partir de ahí la mayor parte de la investigación durante los años 70 se llevó a cabo en Canadá y en Europa. En estas partes del mundo hubo una gran demanda de sistemas de traducción semiautomáticos, debido principalmente al carácter multicultural de estas regiones. En 1976, un grupo de investigadores canadienses obtuvo un gran éxito en la construcción de una sistema de TA llamado "Météo", el cual traducía partes meteorológicos [Tihouin 82]. En el mismo año la Comisión de las Comunidades Europeas (CE) instaló un sistema de TA del francés al inglés llamado Systran [Billmeyer 82]. En años sucesivos se desarrollaron sistemas de TA para otros pares de lenguas europeas. Pocos años después, la comisión europea decidió patrocinar un proyecto ambicioso para desarrollar un sistema de TA plurilingüe para todas las lenguas de la CE, este proyecto fue el proyecto Eurotra [Arnold and des Tombe 82]. El proyecto Eurotra se basó en otros dos proyectos europeos, Ariane [Boitet et al. 82] y SUSY [Maas 77]. Los investigadores de estos proyectos, junto con los que desarrollaron el sistema METAL [Bennett and Slocum 85] llegaron a la conclusión de que el éxito para obtener avances en TA residiría en el desarrollo de sistemas basados en reglas (ver sección 1.4).

En los años 80 creció el interés en sistemas basados en *Interlingua* (ver sección 1.4) aunque se dedicaron más esfuerzos en el desarrollo de técnicas basadas en el conocimiento (knowledge-based), las cuales estaban basadas en el área de los sistemas de comprensión del lenguaje natural. Este área de la inteligencia artificial permitió a los sistemas de TA usar no solamente información lingüística sino también usar contextos de textos y conocimiento acerca del mundo real. Otros dos sistemas holandeses, DLT [Witkam 88] y Rosetta [Rosetta 94] usaron sus propios Interlinguas: el Esperanto y el "Montague semantics", respectivamente.

Al final de los 80 y principios de los 90, los avances en sistemas estadísticos de reconocimiento del habla motivaron el desarrollo de sistemas de TA puramente estadísticos. Como ya hemos comentado, los primeros indicios de trabajo en esta dirección fueron realizados en 1949 cuando Warren Weaver [Brown et al. 93] su-

[1] Del inglés *Fully Automatic High-Quality Translation*

girió técnicas estadísticas y de criptoanálisis para atacar el problema en cuestión. Todos los esfuerzos realizados entonces hacia esta dirección fueron rápidamente abandonados debido a la propia impotencia de los computadores de la época y al gran coste computacional que estos métodos requerían. El tema estuvo aparcado muchos años hasta que en 1988 [Somers 98] durante el segundo congreso del TMI-MT (Theoretical and Methodologies Issues in Machine Translation) en la Universidad Carnegie Mellon, Peter F. Brown de IBM conmocionó a la audiencia presentando una aproximación a la TA completamente nueva, algo que la mayoría de la gente nunca antes había oído e incluso imaginado. Era la aproximación "puramente estadística" de IBM. Esta aproximación estuvo inspirada por el éxito previo que las técnicas estadísticas habían obtenido en el campo del procesamiento del habla. De este modo es como, absteniéndose de la aproximación lingüística racionalista en favor de una aproximación empírica basada en corpus, esta aproximación irrumpió drásticamente en el conocimiento acumulado hasta la fecha de cómo abordar el problema de la traducción automática.

A partir de entonces aparecieron gran cantidad de "nuevas" aproximaciones a la TA, aunque no demasiadas claramente estadísticas como la de IBM, pero todas con un componente común, el uso de un corpus de ejemplos de traducción frente al uso de reglas lingüísticas. Esta aparente diferencia fue a menudo vista como una confrontación, hasta el punto en que en 1992 el congreso del TMI-MT que tuvo lugar en Montreal abordó explícitamente el tema "Empirist vs. Rationalist Methods in MT" [Isabelle 92] (Métodos empiristas frente a racionalistas en traducción automática), cuando para esa fecha la mayoría de los investigadores dedicados al tema ya habían desarrollado sistemas híbridos usando ambas técnicas conjuntamente. Este debate concluyó en considerar todas esas "nuevas" aproximaciones como líneas centrales de investigación en contraste aunque no en conflicto con las viejas técnicas basadas en reglas.

De cualquier modo, el desarrollo más importante en la década pasada fue la aparición de sistemas de TA comerciales. Systran, METAL y otros sistemas pasaron de ser sistemas de ámbito académico a sistemas comerciales utilizables, los cuales fueron incluso comprados por las mismas agencias de gobierno de los EEUU que cancelaron la investigación en los 60. Por la misma época, un gran número de grandes organizaciones (como Ford, Citibank, etc.) desarrollaron sus propios sistemas de TA, o compraron sistemas comerciales desarrollados a medida de acuerdo a sus necesidades. Recientemente, los sistemas de TA han sido incorporados en Internet para facilitar el acceso a documentos plurilingües[2].

[2] Por ejemplo el sistema de Altavista basado en Systran llamado Babelfish, disponible vía `http://balelfish.altavista.com`

1.4 Clasificación de las distintas aproximaciones a la traducción automática

Existen distintos criterios para clasificar los sistemas de TA aunque en la mayoría de los textos actuales [Hobbs 92, Hutchins and Somers 92, Kay et al. 94, Somers 98, Trujillo 99] los sistemas de TA se clasifican de acuerdo a la tecnología o aproximación que siguen. Nosotros expondremos distintas clasificaciones de acuerdo a diferentes criterios.

Una primera clasificación que se puede hacer es teniendo en cuenta el tipo de entrada que aceptan los sistemas de TA. En principio tendremos dos tipos de entrada: texto y habla. La gran diferencia entre estos dos tipos de sistemas es que mientras que en los de entrada de texto las frases a traducir se les supone libre de errores, es decir estarán bien construidas gramaticalmente, en los sistemas con entrada de voz la entrada puede contener errores como pronunciaciones incorrectas o antigramaticales, e incluso podrán producirse errores debidos al propio sistema de RAH. Evidentemente esto conlleva que los sistemas que traten con habla sean más complejos y que la calidad de la traducción de estos sistemas frente a los de entrada de texto sea menor. En esta tesis nos centraremos principalmente en sistemas de traducción con entrada de texto, aunque también daremos algunas ideas de como atacar el problema de la entrada de voz (concretamente en sistemas estadísticos).

Otra posible clasificación sería teniendo en cuenta la aplicación para la que se realiza la traducción. En este caso tenemos tres tipos de sistemas:

- Sistemas de comprensión, también conocidos en la literatura por sistemas *gisting*[3] en donde lo que se pretende con la traducción es obtener una nueva representación de la entrada que de alguna forma contenga el significado de la misma. Un ejemplo típico de este tipo de traducciones es la realización de consultas a una base de datos mediante un interfaz en lenguaje natural, es decir, la orden en lenguaje natural se traduce a un lenguaje de preguntas sobre una base de datos. Otro posible ejemplo sería la traducción de frases en lenguaje natural a un lenguaje de predicados lógicos que representen ciertas características relevantes de la entrada.

- Aplicaciones de post-edición. En este caso se trata de producir una traducción aproximada, la cual será corregida por un traductor humano. Un ejemplo bastante utilizado hoy en día son los sistemas de ayuda a la traducción.

- Sistemas completamente automáticos. En estos el ordenador se utiliza para proporcionar traducciones de alta calidad sin intervención humana. Teniendo en cuenta el estado del arte de la tecnología actual, esto solo es posible de conseguir para:

[3] *Gisting*: extracción de lo esencial

- Tareas de dominios restringidos, es decir que el conjunto de frases que le lleguen al sistema pertenezcan a un dominio de a lo sumo varios miles de palabras. Un ejemplo lo tenemos en el sistema Météo [Tihouin 82] el cual traduce del francés al inglés partes meteorológicos. Dentro de esta categoría podríamos profundizar más en la clasificación de acuerdo a si el dominio de la aplicación es restringido o abierto.
- Lenguajes muy similares (p.e. valenciano o catalán y castellano). Un par de ejemplos son: el traductor SALT [Fabra 99] para el catalán, y el sistema InterNostrum [Canals-Marote et al. 01] para el valenciano.

Otra posible clasificación la podemos ver en la figura 1.1 en donde se muestra el conocido Triángulo de Vauquois de las distintas aproximaciones a la TA. En este triángulo podemos distinguir claramente tres aproximaciones de acuerdo al nivel de análisis (y por consiguiente generación) que realizan. Estas tres aproximaciones las veremos con más detalle en la siguiente clasificación.

Por último podemos realizar una clasificación de acuerdo a la tecnología utilizada para construir los sistemas de TA. En esta clasificación encontramos dos grandes grupos totalmente diferentes y opuestos: los sistemas basados en reglas y los sistemas empíricos o basados en corpus. Veamos con más detalle estos dos grandes grupos.

1.4.1 Sistemas basados en reglas

En los sistemas basados en reglas los expertos establecen una serie de reglas de cómo debe realizarse la traducción. Normalmente la creación de esas reglas requiere de un gran trabajo humano, en el que se necesita el conocimiento de lingüistas expertos en ambos idiomas. Normalmente los sistemas basados en reglas tienen dos componentes comunes: una fase de análisis, que analiza el texto origen, y una fase de síntesis o generación, la cual produce la traducción al idioma destino. Entre estas dos fases puede existir una tercera que consiste en la transferencia que intenta adaptar la fase de análisis a la fase de generación. Esta aproximación sigue exactamente el esquema de la figura 1.1. En la parte más baja de la pirámide vemos lo que se conoce como *Interlingua*, que no es más que una representación obtenida mediante una gran cantidad de análisis y muy buena para realizar una generación directa. Evidentemente no todos los sistema basados en reglas usan un *Interlingua*, ya que ello requiere un análisis completo de la frase de entrada. Para evitar la realización de ese análisis total de la entrada se inserta la fase de transferencia, de modo que a mayor análisis menor transferencia y viceversa.

Veamos con detalle las distintas aproximaciones:

Traducción directa

Esta es la aproximación adoptada en los primeros sistemas de traducción automática (el primer sistema de Systran se basaba en esta aproximación). Estos sistemas se basan en una traducción de palabra a palabra, en donde solamente se realiza análisis morfosintáctico, el cual trata de identificar categorías gramaticales y algunas otras características como el género, número, tiempos verbales, etc., pero excluye cualquier tipo de correspondencia entre palabras o grupos de palabras.

Traducción por transferencia

La idea de esta aproximación es la siguiente: Supongamos que las frases de cualquier idioma se pueden representar mediante un árbol o de alguna otra forma lógica. Entonces un texto en un lenguaje origen se analiza con cierta minuciosidad produciendo una representación lógica para las frases del texto. A continuación las reglas de transferencia son aplicadas a esta representación para producir una representación al nivel correspondiente para el lenguaje destino. El texto entonces es generado en el lenguaje destino.

Un sistema de traducción de francés a inglés podría tener unas reglas de transferencia como la siguiente, para la frase francesa[Hobbs 92]: *la rémunération du temps suplémentaire* tendría la estructura sintáctica: [NP *la rémunération* [PP *du* [NP *temps suplémentaire*]]]. La correspondiente frase inglesa: *overtime pay* tiene la estructura sintáctica: [NP [N *overtime*] *pay*]. Las reglas de transferencia especifican, como fragmentos del árbol sintáctico francés se corresponden con los fragmentos del árbol sintáctico inglés. Las reglas podrían ser expuestas con cualquier generalidad léxica apropiada.

Existen un número de problemas clásicos con esta aproximación, sobre todo cuando los dos lenguajes pueden expresar sintácticamente el mismo concepto de diferentes formas. Por ejemplo, lo que se expresa con el verbo principal en un lenguaje puede ser expresado adverbialmente en el otro.

Como ya hemos comentado existen diferentes niveles de análisis y por tanto de transferencia. En esta aproximación los diferentes niveles de análisis pasan por análisis sintáctico parcial, análisis sintáctico total y análisis semántico (también a diferentes niveles de complejidad).

Teóricamente la parte de análisis de estos sistemas se puede usar también en la fase de generación y viceversa, en cambio, la mayoría de estos sistemas tienen distintas fases de análisis y generación. Ejemplos de sistemas basados en transferencia son Eurotra y METAL [Bennett and Slocum 85].

Interlingua

Esta aproximación es el caso extremo de la aproximación por transferencia, en donde se hace un profundo análisis del texto origen, hasta conseguir una representación conceptual independiente del lenguaje origen (y destino), a esta

representación conceptual se le llama *Interlingua*. Hay dos ventajas en esta aproximación. Primero, un texto a menudo puede ser analizado para conseguir un nivel conceptual que de lugar a traducciones adecuadas. O sea, el texto debe ser entendido antes de que pueda ser traducido. Segundo, sólo necesitamos definir una correspondencia entre cada lenguaje y el *Interlingua*, en lugar de especificar reglas de transferencia para cada par de lenguajes.

Un problema de este esquema es la dificultad de idear un adecuado *Interlingua*, ya que, por ejemplo, una palabra inglesa tan simple y primitiva como *wall*, al traducirla a francés hay que distinguir si hablamos de un muro visto desde dentro o desde fuera.

Al igual que en la aproximación anterior, cualquier componente del *Interlingua* se puede usar tanto en la frase de análisis como en la de generación. Los sistemas DLT [Witkam 88] y Rosetta [Rosetta 94] son ejemplos de sistemas basados en *Interlingua*.

La gran mayoría de los libros de texto dedicados a traducción automática describen ampliamente sistemas basados en reglas [Hobbs 92, Dorr 93, Arnold et al. 94].

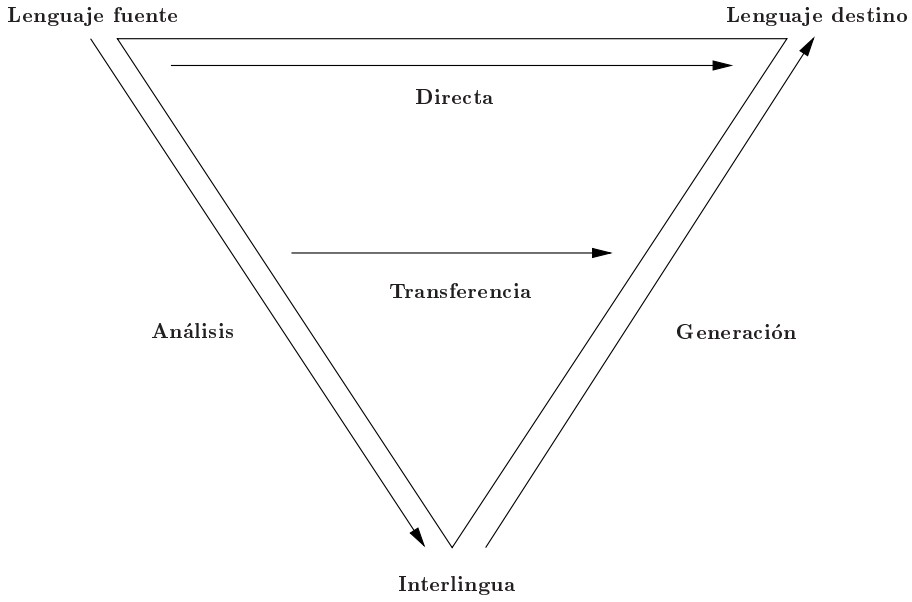


Figura 1.1: El triángulo de Vauquois [Vauquois 75]. Distintas aproximaciones a la TA.

1.4.2 Sistemas basados en corpus

Dentro de esta categoría están todas las que se conocen como aproximaciones empíricas a la TA. Lo que caracteriza a estos sistemas es que toda la información que utilizan para construir sistema de TA son corpus de ejemplos de traducciones entre un par de idiomas, y todo se realiza de forma automática. Una gran ventaja de estos sistema frente a los basados en reglas es que una vez desarrollada la tecnología para un par dado de idiomas, es muy fácil y rápido desarrollar sistemas de traducción para nuevos pares de idiomas (o nuevos dominios entre mismos idiomas) siempre y cuando se disponga de gran cantidad de datos (corpus) de entrenamiento. En la figura 1.2 podemos ver la arquitectura general de un sistema empírico de traducción automática. En los sistemas puramente empíricos, el punto de partida es un conjunto o corpus de pares de frases de entrenamiento, los cuales habrán sido obtenidos por traductores expertos. Con la fase de entrenamiento, se obtienen de forma automática las fuentes de conocimiento en las que se basa el sistema de traducción, en función del corpus de entrenamiento previamente elegido. Esta fase de entrenamiento puede llevar implícita una fase de preprocesado del corpus con la finalidad de facilitar el entrenamiento propiamente dicho y de obtener unas fuentes de conocimiento mejores y más robustas. La regla de decisión deberá combinar de forma óptima todas las fuentes de conocimiento de que dispone el sistema para realizar la traducción de forma óptima. En caso de que en la fase de entrenamiento se haya realizado un preproceso del corpus, adicionalmente, deben existir fases de pre y postproceso, las cuales también facilitarán la tarea de traducción propiamente dicha. El preproceso deberá ser el mismo que el aplicado en al corpus de entrenamiento y el postproceso justo el contrario.

Una aproximación empírica (o basada en corpus) puede seguir tanto una aproximación directa como por transferencia. En este trabajo de tesis trabajaremos con modelos de traducción que esencialmente realizan traducciones palabra a palabra, y por lo tanto siguen una aproximación directa a la TA.

En esta categoría podemos encontrar dos grandes aproximaciones, los sistemas basados en ejemplos (EBMT[4]) y los sistemas estadísticos. En realidad estas dos aproximaciones también se pueden encuadrar dentro de la clasificación de la figura 1.1 dado que la aproximación estadística es un caso especial de traducción directa (traducción palabra a palabra siguiendo ciertas distribuciones de probabilidad) y la traducción basada en ejemplos es un caso particular de la aproximación por transferencia.

Sistemas basados en ejemplos (EBMT)

Diferentes autores han usado nombres alternativos a esta aproximación, quizás por intentar diferenciar de alguna forma sus propias aproximaciones: TA basada en analogía, TA basada en memorización y TA basada en casos han sido

[4] Del inglés *Example-Based Machine Translation*.

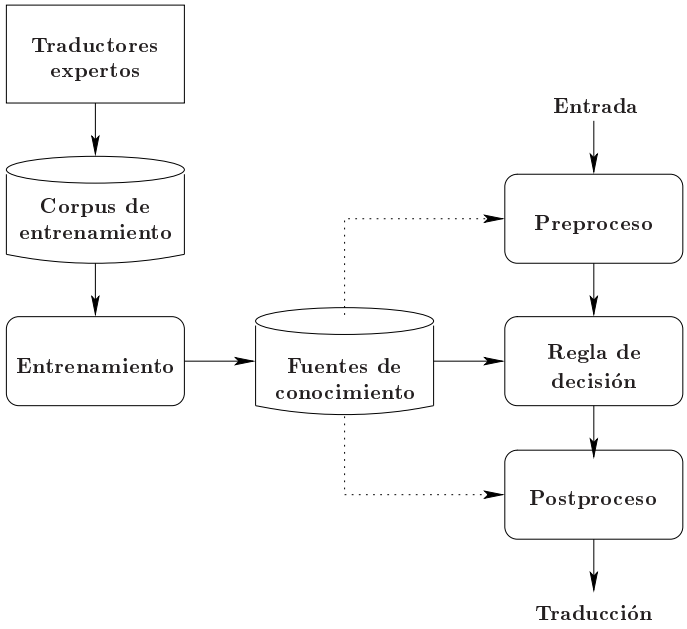


Figura 1.2: Arquitectura general de un sistema empírico de TA.

los nombres mayormente utilizados. En general, esta aproximación hace uso de un corpus o bases de datos de pares de ejemplos de frases traducidas, y el proceso de traducción de una nueva frase se obtiene *comparando* dicha frase con el corpus total extrayendo posibles hipótesis y *combinándolas* de forma apropiada.

La esencia de los sistemas basados en ejemplos se basa principalmente en dos postulados de M. Nagao (el precursor de esta aproximación) que dicen: 1) El hombre no traduce una frase simplemente realizando un profundo análisis lingüístico, y 2) el hombre traduce, primero descomponiendo una frase en ciertos fragmentos, traduce estos fragmentos a otro lenguaje, y finalmente compone adecuadamente estos fragmentos. La traducción de los fragmentos se realiza por un principio de analogía con ejemplos reales dados como referencia [Nagao 84].

Por lo tanto en un sistema basado en ejemplos tenemos tres componentes principales: una comparación de fragmentos con una base de ejemplos reales, una identificación de la traducción de los fragmentos, y una combinación de esos fragmentos traducidos para obtener la frase final. Un amplia descripción de esta metodología se puede encontrar en [Nagao 84, Sato and Nagao 90].

Sistemas estadísticos

Algunos autores como [Somers 98] encuadran a la aproximación estadística dentro de la aproximación basada en ejemplos (EBMT) por el hecho de que

depende de un corpus bilingüe, comentando que el proceso de *comparación-combinación* que caracteriza a los métodos EBMT se implementa de una forma completamente diferente, enfocándose, como es de esperar, en aspectos de estimación de parámetros estadísticos para una serie de modelos de traducción y de lenguaje. Esta gran diferencia es lo que a nosotros y otros autores nos ha llevado a considerarla como una aproximación diferente.

La aproximación estadística fue estudiada en sus inicios en el centro de investigación de IBM en Yorktown Heights. Para trabajar con esta aproximación correctamente debemos tener disponible una gran cantidad de texto con información relevante. Esto nos lleva a tener que simplificar el modelo de los lenguajes y la transferencia entre ellos, hasta el punto en que los análisis estadísticos lleguen a ser factibles.

En los primeros trabajos realizados por Peter F. Brown y sus colaboradores hicieron algunas simplificaciones [Hobbs 92]. En lugar de tener una gramática completa del lenguaje destino, se tiene una gramática simplificada al caso de estudio. En lugar de reglas de transferencia, había, primero un modelo de correspondencias léxicas entre los dos lenguajes, y segundo un modelo de alineamiento entre las frases. El modelo de alineamiento, en lugar de usar la información sobre las correspondencias entre los árboles sintácticos de las reglas de transferencia, recoge una simple relación estructural como la ilustrada en la figura 1.3:

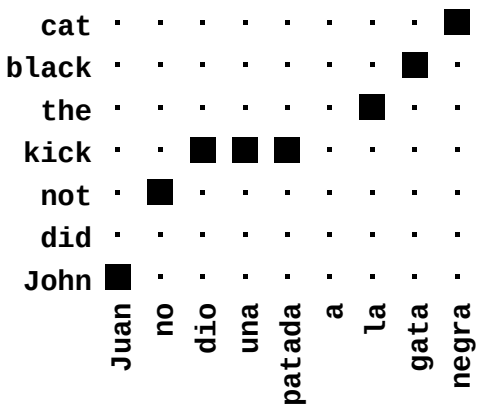


Figura 1.3: Información de alineamiento.

No hay una razón principal por lo que esta aproximación sea tan superficial y de hecho en los trabajos más recientes por el equipo de IBM incorporan más información estructural en sus modelos. Además hoy en día se dispone de gran

cantidad de textos traducidos de un idioma a otro lo que hace posible que se obtengan unos mejores alineamientos entre palabras y conseguir también unas reglas de derivación de forma estadística.

Un problema que nos encontramos al trabajar con textos reales, es el de como analizar frases muy largas. Esto ha sido tratado en los últimos años de una forma especial llegando a la conclusión de que las frases largas se deben romper o descomponer en la medida de lo posible, en frases correctas más pequeñas. Las aproximaciones estadísticas a la traducción realmente no son más que gran cantidad de intensivas computaciones y el hecho de que las frases lleguen a ser muy largas es más importante aquí que en otras aproximaciones. El problema de romper la sentencia en frases para que pueda ser traducida independientemente debe ser hecha mucho antes de abordar el problema de traducción propiamente dicho.

Esta aproximación usualmente se conoce como el sistema *Candide* de IBM [Brown et al. 90, Brown et al. 93], aunque haya sido también adoptada y ampliamente estudiada por otros investigadores [Vogel et al. 96, Tillmann et al. 97a, Tillmann et al. 97b, Wang and Waibel 97, García-Varea and Casacuberta 98, García-Varea et al. 98, Nießen et al. 98, Ney 99, Ney et al. 00], y es la que vamos a abordar en esta tesis con algunas restricciones particulares, de las cuales ya hablaremos más adelante cuando entremos en detalles.

Otros sistemas basados en corpus

Dentro del marco de las aproximaciones basadas en corpus (ejemplos) existen dos aproximaciones distintas que difieren en cierta medida de las anteriores. Una es la conocida como aproximación conexionista (redes neuronales), la cual también se suele enmarcar en el marco de la aproximación estadística. En esta línea no se han realizado muchos trabajos hasta la fecha, cabe destacar entre ellos [Waibel et al. 91, McLean 92, Castaño et al. 97, Koncar and Guthrie 97]. Otra es la conocida como aproximación basada en métodos de estados finitos [Vidal and Prieto 92, Castellanos et al. 94, Llorens et al. 95, Alshawi 96a, Alshawi 96b, Alshawi and Xiang 97, Vidal 97, Knight and Al-Onaizan 98, Amengual et al. 00].

1.5 Aproximación Estadística a la TA

1.5.1 Traducción estadística

Consideremos el problema de traducir una frase $\mathbf{f}$ en un lenguaje de entrada $\mathcal{F}$ a otra frase $\mathbf{e}$ en un lenguaje[5] de salida $\mathcal{E}$. Dada una frase de $\mathcal{F}$, imaginemos que fue originalmente producida por una frase equivalente de $\mathcal{E}$. Para obtener la frase de $\mathcal{F}$, la frase de $\mathcal{E}$ fue transmitida a través de un canal de transmisión con ruido, el cual tiene la curiosa propiedad de que las frases de $\mathcal{E}$ enviadas a ese canal emergen en su salida como la correspondiente traducción en $\mathcal{F}$. Este formalismo se conoce como el *modelo de canal ruidoso*, cuyas características pueden ser determinadas de forma experimental y expresadas matemáticamente.

Este formalismo puede ser explotado para producir traducciones de un lenguaje de entrada a otro de salida del siguiente modo:

Dada una frase $\mathbf{f}$ en un lenguaje de entrada, se pretende traducir a otra frase $\mathbf{e}$ perteneciente a otro lenguaje de salida. De todas las posibles frases en el lenguaje de salida, elegiremos aquella que nos devuelva la mayor probabilidad de acuerdo a la siguiente ecuación: [6]

$$\hat{\mathbf{e}} = \arg\max_{\mathbf{e}} Pr(\mathbf{e}|\mathbf{f}) \tag{1.1}$$

Aplicando la regla de decisión de Bayes y teniendo en cuenta que $Pr(\mathbf{f})$ no depende de $\mathbf{e}$ tenemos que:

$$\hat{\mathbf{e}} = \arg\max_{\mathbf{e}} \{Pr(\mathbf{e}) \cdot Pr(\mathbf{f}|\mathbf{e})\} \tag{1.2}$$

Esta ecuación la podemos ver como una representación del proceso por el cual un humano traduce una frase del lenguaje de entrada al lenguaje de salida, y podemos pensar en ella como en un paseo mental a través de una lista de todas las frases del lenguaje de salida calculando la probabilidad de cada frase $\mathbf{e}$ ($Pr(\mathbf{e})$) y la probabilidad condicionada $Pr(\mathbf{f}|\mathbf{e})$, y eligiendo la mejor de ellas. En definitiva, se trata de elegir la frase $\hat{\mathbf{e}}$ que maximice ese producto de probabilidades.

A la aproximación de la ecuación (1.2) se le conoce aproximación del modelo de la fuente y el canal (ver sección 2.2.2)[Brown et al. 90] o como la ecuación fundamental de la traducción automática estadística [Brown et al. 93], donde: $Pr(\mathbf{e})$ representa la probabilidad de obtener la cadena de salida, y $Pr(\mathbf{f}|\mathbf{e})$ es la probabilidad de obtener $\mathbf{f}$ habiendo observado $\mathbf{e}$. En la práctica se obtiene una estimación de $Pr(\mathbf{e})$ mediante un *modelo de lenguaje* y de $Pr(\mathbf{f}|\mathbf{e})$ mediante un *modelo de traducción*.

[5] Por convenio y similitud a la notación utilizada en [Brown et al. 93] vamos a utilizar $\mathbf{f}$ ($french, foreign, ...$) y $\mathbf{e}$ ($english$) para denotar las frases de entrada y salida respectivamente

[6] El convenio de notación es el siguiente. El símbolo $Pr(\cdot)$ se usa para denotar distribuciones de probabilidad generales sin prácticamente asunción alguna, mientras que para denotar distribuciones de probabilidad basadas en un modelo utilizaremos, dependiendo del modelo al que hagamos referencia, letras minúsculas del modo $p(\cdot)$, $t(\cdot)$, $n(\cdot)$, $d(\cdot)$, etc.

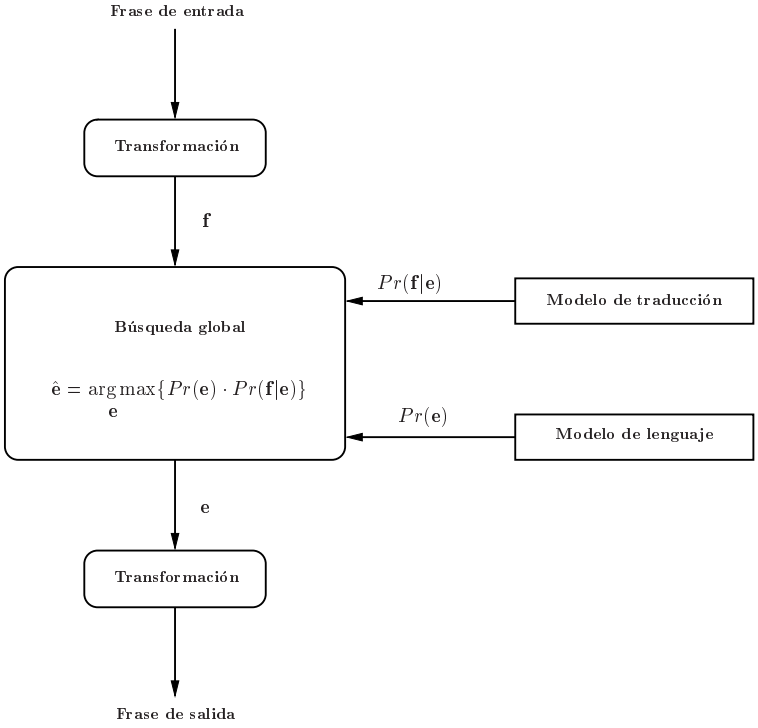


Figura 1.4: Arquitectura general del proceso de traducción basado en la regla de Bayes.

En la figura (1.4)[Ney et al. 00] podemos ver la arquitectura general de la traducción basada en la regla de Bayes. Observando la figura, podemos ver que el proceso de traducción está formado por cuatro partes bien diferenciadas:

- Un modelo de traducción ($Pr(\mathbf{f}|\mathbf{e})$): El modelo de traducción engloba un modelo léxico o diccionario estocástico, el cual nos da la probabilidad de que una palabra e se traduzca por una palabra f, o bien visto desde un punto de vista generativo que la palabra e genere la palabra f. También engloba a un modelo de alineamiento, el cual se define como un mapeo palabra a palabra entre las distintas posiciones de las frases de entrada y salida.

- Un modelo de lenguaje ($Pr(\mathbf{e})$): Este modelo proporciona la probabilidad de que una frase **e** sea generada.

- Un proceso de búsqueda: Esta parte es la que podemos considerar como el proceso de traducción propiamente dicho. Basándose en los modelos

de traducción y lenguaje, el proceso resuelve la ecuación (1.2) obteniendo una hipótesis de la frase de salida $\hat{\mathbf{e}}$ que mejor se adapta a las restricciones impuestas por los modelos.

- Una serie de transformaciones, tanto a la entrada como a la salida, las cuales son útiles para realizar pre y/o post-procesos para facilitar el manejo de las frases y en general para mejorar el rendimiento del sistema.

Normalmente el entrenamiento de los modelos involucrados se realiza siguiendo la aproximación de máxima verosimilitud. Supongamos que el modelo de lenguaje depende de una serie de parámetros γ, o sea $Pr(\mathbf{e}) = p_\gamma(\mathbf{e})$, y que el modelo de traducción depende de una serie de parámetros θ, es decir $Pr(\mathbf{f}|\mathbf{e}) = p_\theta(\mathbf{f} \mid \mathbf{e})$. Entonces el valor óptimo para dichos parámetros se obtendrá maximizando la verosimilitud para un conjunto/corpus paralelo de pares de frases de entrenamiento $\{(\mathbf{f}_s, \mathbf{e}_s) : s = 1, \ldots, S\}$ [Brown et al. 93, Och and Ney 00b] del siguiente modo:

$$\hat{\theta} = \arg\max_{\theta} \left\{ \prod_{s=1}^{S} p_\theta(\mathbf{f}_s \mid \mathbf{e}_s) \right\} \tag{1.3}$$

$$\hat{\gamma} = \arg\max_{\gamma} \left\{ \prod_{s=1}^{S} p_\gamma(\mathbf{e}_s) \right\} \tag{1.4}$$

Normalmente, para evitar que los parámetros de los modelos se ajusten demasiado a los datos de entrenamiento y no se obtenga por tanto una generalización deseable, en la práctica es necesario apartarse de esta estimación por máxima verosimilitud realizando algún tipo de suavizado en los modelos.

Finalmente obtenemos la siguiente regla de decisión:

$$\hat{\mathbf{e}} = \arg\max_{\mathbf{e}} \{ p_{\hat{\gamma}}(\mathbf{e}) \cdot p_{\hat{\theta}}(\mathbf{f} \mid \mathbf{e}) \} \tag{1.5}$$

La gran mayoría de los sistemas estadísticos de traducción encontrados en la bibliografía siguen esta aproximación.

1.5.2 Estado del arte

En esta sección hemos hecho una recopilación de los trabajos más relevantes hasta la fecha en cuanto a lo que se refiere a la aproximación estadística a la traducción automática propiamente dicha. Evidentemente existen muchos más trabajos relacionados con el tema que los aquí expuestos, de modo que éstos más específicos irán siendo citados durante el desarrollo de la tesis dentro del contexto que están directamente relacionados. A título de ejemplo, los trabajos relacionados con el tema de máxima entropía estarán comentados en el tema que hemos dedicado expresamente a ello.

Como hemos comentado en la sección de introducción, prácticamente todo el trabajo de importancia relevante dentro del marco de la aproximación puramente estadística ha sido desarrollado durante la década de los noventa.

El disparo de salida de la aproximación estadística a la traducción automática lo dieron Peter F. Brown y sus colegas del centro Thomas J. Watson de IBM. En [Brown et al. 90] se da una introducción al tema, en [Brown et al. 91b] se estudian ciertas técnicas para el alineamiento de frases en corpus bilingües. Los mismos autores en [Brown et al. 93] exponen los famosos 5 modelos de traducción de IBM, con todo el desarrollo matemático para la estimación por máxima verosimilitud de los parámetros de los modelos utilizando el algoritmo EM (Expectation-Maximization). Existen otros trabajos de este grupo de investigación relacionados con el tema como son: [Brown et al. 92] en donde exponen un modelo de lenguaje de n-gramas basado en clases y [Brown et al. 94] en el que se habla de reconocimiento del habla aplicado a temas de traducción. En [Berger et al. 94] se describe globalmente el sistema "Candide" de IBM que consiste en un sistema automático de traducción texto a texto del francés al inglés. Este sistema, partiendo del formalismo del modelo de canal ruidoso, desarrolla una serie de modelos paramétricos probabilísticos para modelos de lenguaje y de traducción, así como algoritmos numéricos para entrenar dichos modelos a partir de ejemplos. En definitiva, el sistema "Candide" es el resultado de todo el trabajo realizado por este grupo en esta línea de investigación. En estos trabajos el problema de la búsqueda se soluciona mediante costosos algoritmos basados en la técnica de ramificación y poda.

Otro grupo de investigadores dedicados al tema es el de Hermann Ney en el RWTH de la Universidad Tecnológica de Aachen. De los trabajos realizados por este grupo podemos destacar: [Nießen et al. 98] donde presentan un algoritmo de búsqueda para un modelo de traducción basado en lo que ellos llaman alineamientos invertidos; en [Tillmann et al. 97a], en [Tillmann et al. 97b] y en [Vogel et al. 96] presentan otro modelo de traducción basado en alineamientos monótonos del mismo modo que se utilizan los MOM (modelos ocultos de Markov) en reconocimiento del habla, también presentan un algoritmo de búsqueda y su versión mejorada. En todos estos trabajos podemos destacar el uso de técnicas de programación dinámica en los procesos de búsqueda. En [Ney et al. 00] podemos encontrar un resumen de todas las técnicas desarrolladas por este grupo hasta la fecha.

En otros trabajos como [Melamed 97] se describe un modelo de traducción palabra a palabra. En [Wang and Waibel 97] se presenta un algoritmo de decodificación/búsqueda conocido como "stack decoding", y en [Wu 96] podemos encontrar un algoritmo de búsqueda en tiempo polinómico.

Otro grupo de investigación dedicado a temas de traducción automática tanto de texto como voz es el grupo PRHLT (Pattern Recognition and Human Langauge Technology) del DSIC (Depto. de Sistemas Informáticos y Computación) y del ITI (Instituto Tecnológico de Informática) en la Universidad Politécnica de Valencia, en el que su mayor trabajo se centra en el desarrollo de técnicas

de estados finitos para el procesamiento, reconocimiento y traducción del habla. En cuanto a temas estadísticos también se han realizado trabajos de relativa importancia. Concretamente tres tesis doctorales realizadas en el grupo tratan distintos problemas en este ámbito. En [Prat 98] se presentan distintos modelos estocásticos basados en asociación de gramáticas y sus respectivos algoritmos de búsqueda; en[Prat 01] se presentan algunas mejoras. En [Vilar 98] se presenta un modelo estadístico recursivo adecuado para la segmentación automática de pares de frases cuyo resultado es directamente aplicable a mejorar sistemas de estados finitos como en [Casacuberta 00]. También se han desarrollado modelos conexionistas para tratar el tema en cuestión, concretamente en [Castaño et al. 97] y en [Prat 98].

Cabe destacar el trabajo realizado por una serie de investigadores de todo el mundo, dirigidos por Kevin Knight durante el transcurso de un taller de trabajo organizado por el CLSP (Center for Language and Speech Processing) de la Universidad Johns Hopkins durante el verano de 1999 [Al-Onaizan et al. 99]. En este taller titulado "Traducción automática estadística" se desarrollaron nuevas implementaciones de los modelos 1-4 de IBM, así como una serie de herramientas de pre y post-procesamiento de corpus y una serie de aplicaciones para visualizar los alineamientos entre frases obtenidos por estos modelos de traducción. El paquete que incluye todas estas herramientas se denomina EGYPT y es de libre distribución y uso. Este trabajo ha sido continuado hasta la fecha por parte de los participantes iniciales para desarrollar algunos algoritmos de búsqueda para los modelos implementados anteriormente y una implementación mejorada de los algoritmos de entrenamiento para los modelos de traducción [Och et al. 99], incluyendo una implementación para el modelo 5. Este último trabajo a dado lugar al paquete GIZA++ [Och 00].

En cuanto a la integración de voz en sistemas de traducción estadísticos no hay mucho publicado acerca del tema. Destacar en este sentido algunos trabajos como [Ney 99] donde se muestra una primera aproximación al problema; o en [García-Varea et al. 00] donde se presenta una versión semi acoplada donde de forma iterativa se mezclan los procesos de reconocimiento y traducción para mejorar el rendimiento y en [García-Varea et al. 99] donde se da una versión acoplada utilizando una representación en forma de grafos de palabras como entrada de voz. En el apéndice A se presenta un esbozo de cómo atacar el problema de la entrada de voz en sistemas de traducción automática estadística.

1.6 Objetivos de la tesis

A continuación describiremos cuales han sido los objetivos que, a lo largo de los años dedicados a la investigación en este tema, nos hemos ido marcando para desarrollar este trabajo de tesis. Estos se pueden resumir en tres grandes objetivos:

1. En primer lugar, y que a nuestro juicio todo trabajo de investigación debe

de plantearse, es recopilar información acerca del tema, instruirse en el mismo utilizando los medios de que se dispongan, y establecer una línea por la que encauzar el trabajo a realizar. En definitiva, conocer el estado del arte del tema a tratar.

2. La traducción automática estadística es una disciplina relativamente joven, se podría decir que apenas tiene una década de historia. Uno de los principales problemas que se le plantean es la de desarrollar eficientes y eficaces algoritmos para resolver el problema de la búsqueda. Aunque como comentaremos más adelante este problema es NP-completo nos deberemos de conformar con encontrar algoritmos subóptimos para resolver dicho problema.

   Concretamente, hemos pretendido abordar el problema de la búsqueda desde las perspectivas clásicas para el diseño e implementación de algoritmos de búsqueda, que son:

   - La técnica de programación dinámica.
   - La técnica de ramificación y poda.
   - Y la técnica devoradora.

   Por tanto uno de los objetivos principales de esta tesis ha sido el llevar a cabo el estudio, desarrollo, implementación, experimentación y comparación de algoritmos de búsqueda basados en estas técnicas para resolver el problema de la búsqueda en la traducción automática estadística.

3. Otro de los objetivos ha sido el de intentar de dar mayor versatilidad a los modelos estadísticos de traducción clásicos, (basados en relaciones estructurales palabra a palabras), es decir, dotar a dichos modelos en la medida de lo posible de cierta sintaxis y semántica. Nuestra intención no ha pasado por intentar desviar el enfoque a una aproximación más racionalista al problema sino todo lo contrario, intentar mantener el enfoque empirista. Para ello hemos utilizado el marco que la máxima entropía nos ofrece.

   Por tanto, nuestro tercer principal objetivo ha sido el de estudiar, definir, implementar e integrar dentro de los modelos estadísticos de traducción clásicos, modelos contextuales basados en máxima entropía. Del mismo modo hemos pretendido llevar un estudio experimental del potencial de estos modelos contextuales dentro de sistemas de traducción automática estadística.

Por último, podríamos considerar como objetivo adicional a este trabajo, y ya un poco fuera de lo que a investigación propiamente dicha se trata, es el de proporcionar un documento de referencia a aquellos investigadores que se inicien en el tema. Con esta idea se elaborado esta memoria de tesis, en la que no solamente se han descrito las técnicas, algoritmos, etc. investigados,

sino que además se ha intentado dar una introducción lo más completa posible a la traducción automática estadística. Esto viene a decir que algunos de los capítulos, concretamente la primera parte de la tesis, se podría haber resumido considerablemente. Nosotros hemos querido ir poco más allá, de modo que este documento sea autocontenido en su mayoría. El principal motivo que nos ha inducido a ello ha sido el de aliviar en la medida de lo posible la barrera matemática, que la aproximación estadística a la traducción automática conlleva, que tanto asusta a muchos investigadores que se inician en el tema (sobre todo a expertos en el campo de la lingüística). De este modo, hemos intentado ser lo más intuitivos posibles sin por ello perder rigurosidad en la exposición.

Capítulo 2

Traducción automática estadística. Preliminares

ESTE tema lo hemos dedicado para introducir una serie de conocimientos básicos acerca de la traducción automática estadística. Concretamente hemos desarrollado con cierto nivel de detalle cada uno de los componentes básicos de un sistema de traducción automática estadística (TAE). Para una exposición más rigurosa (sobre todo de los modelos de traducción de IBM [Brown et al. 93]) hemos dedicado el siguiente capítulo, de modo que el lector que esté familiarizado con el tema puede pasar por alto la lectura de este capítulo, exceptuando la sección 2.1 de notación en el que se describe la terminología a utilizar en el resto de la tesis, y la sección 2.5.1 donde se expone el concepto de modelo de error necesario para entender algunos comentarios expuestos a lo largo del resto de capítulos.

2.1 Notación

En este apartado se describe el significado de todos los símbolos matemáticos que se van a utilizar en toda la tesis, de modo que aconsejamos al lector tenga a mano este apartado para identificar fácilmente en todo momento el significado de las expresiones que vayan apareciendo en el texto.

Por similitud con la exposición de [Brown et al. 93] vamos a denotar al lenguaje de entrada con la letra $\mathcal{F}$ y al de salida/destino con la letra $\mathcal{E}$.[1]

[1] De hecho en el papel seminal de Brown y colaboradores [Brown et al. 93] se utilizan estas

$\$$	delimitador de frases
e	una palabra destino
ze	una palabra destino con fertilidad cero
I	longitud de una frase destino
$\mathbf{e} = e_1^I$	una frase origen de longitud I
i	una posición en la frase destino $i = 0, 1, \cdots, I$
$\mathcal{E}$	vocabulario del lenguaje destino
e_i	i-ésima palabra de la frase $\mathbf{e}$
e_0	la palabra vacía/nula de la frase $\mathbf{e}$
e_1^i	subsecuencia $e_1 e_2 \cdots e_i$ de la frase $\mathbf{e}$
$\mathcal{E}_c(e)$	clase a la que pertenece la palabra e
f	una palabra origen
$\mathcal{F}$	vocabulario del lenguaje origen
J	longitud de una frase origen
$\mathbf{f} = f_1^J$	una frase origen de longitud J
j	una posición en la frase origen $j = 1, \cdots, J$
f_j	j-ésima palabra de la frase $\mathbf{f}$
f_1^j	subsecuencia $f_1 f_2 \cdots f_j$ de la frase $\mathbf{f}$
$\mathcal{F}_c(f)$	clase a la que pertenece la palabra f
$\mathbf{a} = a_1^J$	un alineamiento entre $\mathbf{f}$ y $\mathbf{e}$
a_j	posición en $\mathbf{e}$ alineada/conectada con la posición j de $\mathbf{f}$ en el alineamiento $\mathbf{a}$
ϕ_i	número de posiciones de $\mathbf{f}$ alineadas a la posición i de $\mathbf{e}$ (o fertilidad de la palabra e_i)
ϕ_1^i	$\phi_1 \phi_2 \cdots \phi_i$
ϕ_0	fertilidad de la palabra nula (e_0)
$t(f \mid e)$	modelo léxico: probabilidad de traducción de e a f (modelos 1-5)
$n(\phi \mid e_i)$	modelo de fertilidades: probabilidad de que la palabra e tenga una fertilidad de ϕ palabras (modelos 3-5)
$a(j \mid i, J, I)$	modelo de alineamiento: probabilidad de alineamiento de la palabra en la $\mathbf{e}$ con la palabra en la posición j de $\mathbf{f}$ posición i de (modelo 2)
$d(i \mid j, J, I)$	modelo de distorsión: probabilidad de alineamiento inverso (modelo 3)
τ_{ik}	k-ésima palabra origen alineada con e_i
π_{ik}	posición que τ_{ik} ocupa en $\mathbf{f}$
ρ_i	posición de la primera palabra de fertilidad mayor que cero a la izquierda de e_i
$d_{=1}(\Delta_j \mid \cdot, \cdot)$	modelo de distorsión (modelos 4-5)
$d_{>1}(\Delta_j \mid \cdot)$	modelo de distorsión (modelos 4-5)

Tabla 2.1: Tabla de notación

2.2 Algunos conceptos previos

2.2.1 Corpus alineados

Como hemos visto en el tema de introducción la aproximación estadística a la traducción automática está dentro de las aproximaciones basadas en corpus.

Desde el punto de vista de la traducción automática un corpus será una recopilación de información bilingüe organizada de cierta manera. La forma en que esté organizado el corpus es de vital importancia dentro de los sistemas estadísticos de traducción, pues en pocas palabras todos ellos se basan en relaciones estructurales entre las frases de los idiomas origen y destino de la traducción. Es decir, cualquier sistema de traducción automática estadística partirá de la base de un corpus alineado a nivel de frase. A este tipo de corpus se les llama corpus paralelos.

Evidentemente las fuentes de información bilingüe reales de que se dispone, como por ejemplo manuales de usuario, guías de todo tipo, periódicos, boletines oficiales, sitios web plurilingües, etc., estarán alineados a nivel de documento, o como mucho a nivel de párrafo. Por este motivo, siempre es necesario un preproceso de los datos para obtener corpus paralelos, que hagan factible y faciliten la construcción de sistemas estadísticos de traducción. En lo que sigue nosotros supondremos que los corpus de experimentación (entrenamiento, desarrollo, validación y test) estarán alineados a nivel de frase, es decir no desarrollaremos ninguna técnica para realizar este trabajo. De cualquier forma y a título de información existen gran cantidad de trabajos enfocados a este problema, entre ellos podríamos destacar [Brown et al. 91b, Fung and McKeown 94, Fung and Church 94, Papageorgiou et al. 94, Wu 94, Wang et al. 02, Tz-Liang and Su 02].

2.2.2 El modelo de la fuente y el canal

Como hemos comentado en el tema anterior, el objetivo de la TAE es: dada un frase $\mathbf{f}$, buscamos una frase $\mathbf{e}$ que maximice la probabilidad $Pr(\mathbf{e}|\mathbf{f})$(la traducción "más probable"). Normalmente esto se escribe como:

$$\arg\max_{\mathbf{e}} Pr(\mathbf{e}|\mathbf{f})$$

Donde arg max se lee como: "la frase $\mathbf{e}$, de todas las posibles, que produce el máximo valor para $Pr(\mathbf{e}|\mathbf{f})$".

Aplicando la regla de Bayes podemos reescribir la expresión anterior del siguiente modo:

$$\arg\max_{\mathbf{e}} Pr(\mathbf{e}|\mathbf{f}) = \arg\max_{\mathbf{e}} \{Pr(\mathbf{e}) \cdot Pr(\mathbf{f}|\mathbf{e})\}$$

letras pues estos autores desarrollaron un sistema de traducción automática del francés al inglés. Esta notación se utiliza en la mayoría de los trabajos publicados relacionados con el tema a pesar de que los lenguajes origen y destino no sean el francés y/o el inglés.

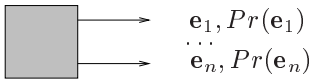
dado que $Pr(\mathbf{f})$ no depende de $\mathbf{e}$ en la maximización.

Esta maximización viene a decir que la traducción más probable $\mathbf{e}$ maximiza el producto de dos términos, (1) la probabilidad de que alguien diga $\mathbf{e}$ en primer lugar, y (2) si ese alguien dijo $\mathbf{e}$, la probabilidad de que alguien más la tradujese a $\mathbf{f}$.

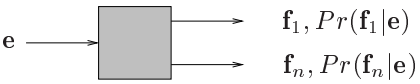
El formalismo del modelo de la fuente y el canal trabaja de esta forma. Imaginemos que alguien tiene $\mathbf{e}$ en mente, pero en el momento de ser escrita se corrompe mediante "ruido" y se convierte en $\mathbf{f}$. Entonces, para recuperar la frase $\mathbf{e}$, traducción más probable de $\mathbf{f}$, razonamos de acuerdo a dos fenómenos: (1) qué tipo de cosas dice la gente en el lenguaje destino $\mathcal{E}$ (o dicho de otra forma, cómo se generan frases en un lenguaje $\mathcal{E}$), y (2) cómo las frases de $\mathcal{E}$ se convierten al lenguaje origen $\mathcal{F}$ (o lo que es lo mismo, cómo se traducen esas frases a otro lenguaje $\mathcal{F}$). A estos dos fenómenos a veces se les denomina "modelado de la fuente" y "modelado del canal". En general, la metáfora del modelo de la fuente y el canal se usa en gran cantidad de problemas de ingeniería, como por ejemplo la transmisión en líneas telefónicas.

Desde un punto de vista informático, estos modelos podríamos entenderlos en términos de sendos programas:

- El primero que modele la fuente o $Pr(\mathbf{e})$, que iría generando frases en el lenguaje $\mathcal{E}$ con sus probabilidades asociadas, o también en un programa que directamente devuelva/genere una lista de frases $\mathbf{e}_i$ con sus correspondientes probabilidades $Pr(\mathbf{e}_i)$, es decir:



- Y el segundo que modele el canal o $Pr(\mathbf{f}|\mathbf{e})$, que iría generando traducciones a $\mathcal{F}$ también con las correspondientes probabilidades, o del mismo modo, un programa que tome una frase $\mathbf{e}$ y devuelva varias frases $\mathbf{f}_i$ junto con sus correspondientes probabilidades $Pr(\mathbf{f}_i \mid \mathbf{e})$, es decir:



Podemos poner los modelos de la "fuente" y del "canal" conjuntamente de la forma:

Como podemos ver, según estos programas, hay muchas más formas de obtener la misma frase $\mathbf{f}$. Cada forma corresponde a las diferentes elecciones de la frase $\mathbf{e}$. Si nos fijamos en los programas (módulos) anteriores todas las flechas van de izquierda a derecha. A esto se le conoce como modelo generativo porque es una teoría de cómo se genera una frase $\mathbf{f}$. La teoría es, primero se genera una frase $\mathbf{e}$, después se convierte a $\mathbf{f}$.

2.2.3 Razonamiento basado en la regla de Bayes

En los programas de la sección anterior todas las flechas apuntan hacia la derecha, es decir, el razonamiento parte de primero generar una frase $\mathbf{e}$, y después convertirla a $\mathbf{f}$. En realidad, a nosotros nos interesa justo lo contrario, traducir frases de un lenguaje origen $\mathcal{F}$ a otro lenguaje destino $\mathcal{E}$. Veamos el porqué del interés de este razonamiento frente al razonamiento directo (es decir, modelar directamente $Pr(\mathbf{e}|\mathbf{f})$). Veámoslo utilizando un símil sobre enfermedades y síntomas.

De forma intuitiva podemos pensar en $\mathbf{f}$ como en un conjunto de síntomas médicos, y en $\mathbf{e}$ una enfermedad en concreto. Desde un punto de vista médico sería interesante poder diagnosticar una enfermedad a partir de unos síntomas. Es decir, sería interesante poder construir un modelo para $Pr(\mathbf{e}|\mathbf{f})$, pero esto no es nada fácil pues en principio, hay gran cantidad de enfermedades que podrían provocar los mismos síntomas. En cambio, dado que los biólogos saben con cierta certeza qué síntomas son generados por ciertas enfermedades, sería factible construir un modelo para $Pr(\mathbf{f}|\mathbf{e})$. Por otra parte también podemos razonar con cierta certeza acerca de $Pr(\mathbf{e})$ (cómo de rara o con qué frecuencia se da una cierta enfermedad), utilizando fuentes de información independientes como lo son los archivos de los hospitales.

Por tanto, para solucionar el problema inicial $Pr(\mathbf{e}|\mathbf{f})$, podemos creamos un modelo generativo de modo que podemos razonar acerca de la probabilidad de que una concreta enfermedad $\mathbf{e}$ ocurra, así como en la probabilidad de que los síntomas $\mathbf{f}$ sean provocados por esa enfermedad $\mathbf{e}$ en particular. Esto es, estaríamos modelando por un lado $Pr(\mathbf{e})$ y por otro $Pr(\mathbf{f}|\mathbf{e})$.

En general, estas dos probabilidades pueden discrepar en la medida de que podría haber una enfermedad común que a menudo provoque los síntomas $\mathbf{f}$, y podría haber una enfermedad muy rara que por contra siempre provoque esos síntomas.

Desde un punto de vista lingüístico no parece que sea tan obvia la complejidad de construir un modelo para $Pr(\mathbf{f}|\mathbf{e})$ o para $Pr(\mathbf{e}|\mathbf{f})$, puesto que ambos se podrían construir a partir de un mismo corpus paralelo de traducciones de un lenguaje a otro. La clave del tema está en que utilizando el razonamiento según la regla de Bayes incluimos una fuente de información adicional e independiente, $Pr(\mathbf{e})$, la cual en este ámbito es de gran ayuda pues se podría obtener a partir de un corpus independiente mucho mayor que el utilizado para obtener $Pr(\mathbf{f}|\mathbf{e})$ o $Pr(\mathbf{e}|\mathbf{f})$. Es decir, por muy grande que podamos encontrar un corpus bilingüe siempre será más fácil encontrar uno monolingüe que podamos utilizar como información adicional. Entraremos un poco más en detalle en estos aspectos en las dos secciones siguientes.

2.2.4 Reordenado de palabras en traducción

Si razonamos directamente sobre la traducción usando $Pr(\mathbf{e}|\mathbf{f})$, las probabilidades asociadas a los parámetros del modelo deberían ser muy buenas. Pero por otra parte, si partimos del razonamiento utilizando la regla de Bayes, teóricamente podríamos obtener buenas traducciones incluso en el caso en que las probabilidades $Pr(\mathbf{f}|\mathbf{e})$ no sean tan precisas como las anteriores.

Por ejemplo, supongamos que asignamos un gran valor a $Pr(\mathbf{f}|\mathbf{e})$ solo si las palabras de $\mathbf{f}$ son en general traducciones de las palabras de $\mathbf{e}$. Las palabras de $\mathbf{f}$ podrían estar en cualquier orden (en principio eso no importa), de modo que no estaríamos hablando de un buen modelo de como convertir frases de $\mathcal{F}$ a frases de $\mathcal{E}$, sino más bien de un modelo de como convertir frases de $\mathcal{F}$ a frases no precisamente correctas de $\mathcal{E}$.

Centrémonos ahora en $Pr(\mathbf{e})$. Supongamos que asignamos un alto valor a $Pr(\mathbf{e})$ sólo si $\mathbf{e}$ es gramaticalmente correcta. Esto es bastante razonable, aunque bastante difícil en la práctica.

Si utilizamos ambas fuentes de información ($Pr(\mathbf{e})$ y $Pr(\mathbf{f}|\mathbf{e})$) en el proceso de convertir la frase observada $\mathbf{f}$ (p.e. *la casa azul*) a su traducción $\mathbf{e}$ más probable, cada posible $\mathbf{e}$ tendrá una probabilidad o *puntuación*[2]) $\{Pr(\mathbf{e}) \cdot Pr(\mathbf{f}|\mathbf{e})\}$ asociada. El factor $Pr(\mathbf{f}|\mathbf{e})$ asegurará que una buena hipótesis $\mathbf{e}$ tendrá palabras que en general serán traducciones de las palabras de $\mathbf{f}$ (en nuestro caso, las frases *the blue house* y *house blue the* serán buenas candidatas si traducimos al inglés). En cambio, es evidente que algunos órdenes en las palabras serán gramaticalmente correctos mientras que otros no, de modo que el factor $Pr(\mathbf{e})$ decrementará la *puntuación* de las frases no gramaticales, e incrementará el de las gramaticales.

En efecto, $Pr(\mathbf{e})$ se preocupa por el orden de las palabras de la frase de $\mathcal{E}$ mientras que $Pr(\mathbf{f}|\mathbf{e})$ no. Esto hace que $Pr(\mathbf{f}|\mathbf{e})$ sea más fácil de construir de lo que podríamos pensar. Este modelo solo necesita saber si un conjunto de palabras (frase) de $\mathcal{E}$ corresponden a un conjunto de palabras (frase) de $\mathcal{F}$[3]. Esto se podría hacer simplemente utilizando un diccionario bilingüe, o de forma más algorítmica sería un módulo que sea capaz de convertir un conjunto de palabras en otro y asignarle una *puntuación* de $Pr(\mathbf{f}|\mathbf{e})$ al par de conjuntos (frases).

De cualquier forma si intentásemos poner en orden la siguiente frase *las de son tipo y respectiva en es rutina ordinaria una computadora programación la normal del recursión ecuaciones*, nos podríamos hacer las siguientes preguntas: ¿Qué tipo de conocimiento estaríamos aplicando?, ¿sería una máquina capaz de hacerlo?,

[2] Dado que en general el cálculo de $Pr(\mathbf{e})$ o $Pr(\mathbf{f}|\mathbf{e})$ conlleva la multiplicación de gran cantidad de valores de probabilidad (entre 0.0 y 1.0), esto puede dar lugar a valores infinitamente pequeños. Por este motivo, en la práctica, se utilizan logaritmos de probabilidades para evitar desbordamientos. Por ello en lo sucesivo nos referiremos indistintamente a los conceptos de *puntuación* y probabilidad asociados a una distribución de probabilidad, aunque ello suponga abusar del lenguaje.

[3] A este tipo de modelos se les conoce como *bag of words models*.

¿podemos pensar en un modo automático de comprobar cómo de bien lo haría esa máquina sin la necesidad de la ayuda de un humano?

Pensemos ahora en ordenar la siguiente frase: *a ama Juan María*. Esto último sería verdaderamente difícil. Parece ser que después de todo $Pr(\mathbf{f}|\mathbf{e})$ necesitaría saber acerca del orden las palabras, no solamente proporcionar un conjunto de palabras sin más. Pero quizás simplemente con saber un poco acerca del orden de las palabras sería suficiente.

2.2.5 Elección de palabras en traducción

Del mismo modo que hemos comentado que $Pr(\mathbf{f}|\mathbf{e})$ debería, en cierta medida, tener información acerca del orden de las palabras a generar, el modelo $Pr(\mathbf{e})$ podría ser también útil para seleccionar traducciones entre palabras. Por ejemplo, supongamos que existe una palabra en castellano que se traduce al inglés bien por *in* o por *on*. Entonces habría dos frases en inglés igualmente probables con igual *puntuación* $Pr(\mathbf{f}|\mathbf{e})$, por ejemplo: (1) *she is in the end zone*, (2) *she is on the end zone* (evidentemente ignorando otras posibles traducciones como: *zone end the in is she*, la cual probablemente también tendría una buena *puntuación* $Pr(\mathbf{f}|\mathbf{e})$). Bien, la primera frase estará escrita en mucho mejor inglés que la segunda con lo que tendrá una *puntuación* $Pr(\mathbf{e})$ mucho mejor, y por lo tanto una mejor *puntuación total* $\{Pr(\mathbf{e}) \cdot Pr(\mathbf{f}|\mathbf{e})\}$.

En este punto nos surge una pregunta, ¿de dónde vienen o cómo podemos obtener esas probabilidades o en general distribuciones de probabilidad? En las dos siguientes secciones intentaremos responder a esta pregunta.

2.3 Modelos de lenguaje

Aunque no es uno de los objetivos de esta tesis el hacer un estudio de modelos de lenguaje, por completitud en la exposición y dado que (como hemos visto en el tema de introducción) el modelo de lenguaje es una de las partes importantes dentro de un sistema de TAE, vamos a dedicar esta sección a ello comentando los aspectos más relevantes.

Un modelo de lenguaje $Pr(\mathbf{e})$ debe ser capaz de dar una idea de lo correcta (de acuerdo a un determinado criterio sintáctico, semántico, gramatical, etc.) que sea una frase $\mathbf{e}$ generada en un lenguaje destino.

Evidentemente lo interesante será hacerlo de forma automática, es decir deseamos crear una máquina que sea capaz de asignar una probabilidad $Pr(\mathbf{e})$ a cada frase $\mathbf{e}$ de nuestro lenguaje de salida.

Una primera idea de como solventar este problema podría ser construir un programa que sepa mucho acerca del mundo, acerca del tipo de cosas que la gente suele decir o discutir, acerca de las estructuras gramaticales que la gente usa cuando describe ciertos eventos u objetos, etc. Podríamos por tanto escribir

un programa que contemple todas estas posibilidades para obtener el resultado deseado.

Otra idea más simple sería almacenar cada frase que cualquiera pueda decir en nuestro lenguaje de salida. Supongamos que recopilamos un millón de frases. Entonces si la frase *¿Cómo están ustedes?* aparece 7641 veces en la base de datos, entonces diremos que $Pr(\text{¿Cómo están ustedes?}) = 7641/1000000 = 0.007641$.

Un gran problema de esta aproximación es que frases perfectamente correctas podrían tener un valor de $Pr(\mathbf{e})$ igual a cero debido a que no hayan sido vistas anteriormente. De modo que aunque tuviéramos un buen programa para calcular $Pr(\mathbf{f}|\mathbf{e})$, no serviría de nada si la $Pr(\mathbf{e})$ asociada fuese siempre cero. Esto es tanto como decir que dos frases completamente distintas serán consideradas traducciones igualmente probables.

En general somos capaces de juzgar si una frase es gramaticalmente correcta o no sin necesidad de almacenar una gran base de datos de frases. Por ejemplo, casi con toda seguridad que la frase *Nunca es tarde si la sopa es buena*, nunca antes la hemos oído o visto escrita[4], en cambio somos capaces de decir que la frase es gramaticalmente correcta. Entonces parece que intuitivamente seamos capaces de trocear la frase y si sus componentes son buenos, y los podemos combinar de forma razonable, entonces podríamos decir que la frase es correcta.

2.3.1 Modelos de n-gramas

Para los computadores la forma más fácil de trocear una frase en componentes es considerar subcadenas. A una subcadena de n palabras se le llama n-grama. Si $n = 2$, hablaríamos de bigramas, si $n = 3$ estaríamos hablando de trigramas y si $n = 1$ de unigramas (o simplemente palabras).

Si una frase está formada por una serie de n-gramas razonables, entonces probablemente será una frase razonable.

En general y más formalmente, un modelo de n-gramas se define como:

$$Pr(e_1^I) = \prod_{i=1}^{I+1} p(e_i \mid e_1, \cdots, e_i) \tag{2.1}$$

donde se asume que $e_0 = e_{-1} = \cdots = e_{-n+1} = e_{I+1} = \$$ es un símbolo para delimitar frases.

Típicamente, en la mayoría de los sistemas estadísticos de traducción automática se utilizan trigramas [Ney et al. 95], de modo que un modelo de trigramas, con una serie de parámetros γ vendría dado por:

$$p_\gamma(e_1^I) = \prod_{i=1}^{I+1} p(e_i \mid e_{i-2}, e_{i-1}) \tag{2.2}$$

[4] Excepto los aficionados a las greguerías - `http://www.elhuevodechocolate.com/greguerias.htm`

Como hemos comentado en el tema de introducción la estimación de los parámetros para un modelo de n-gramas se realiza por máxima verosimilitud a partir de un corpus de entrenamiento. Como podemos ver en [Ney et al. 97], la frecuencia con que aparece un determinado n-grama con respecto a su historia es un estimador máximo verosímil para un modelo de n-gramas. Es decir que a partir de un corpus $\mathcal{C}$ de entrenamiento podemos estimar los valores de todos los parámetros del modelo simplemente calculando:

$$p(e_i \mid e_{i-2}, e_{i-1}) = \frac{N(e_{i-2}, e_{i-1}, e_i)}{N(e_{i-2}, e_{i-1})} \tag{2.3}$$

donde $N(e_{i-2}, e_{i-1}, e_i)$ es el número de veces que la secuencia de palabras (e_{i-2}, e_{i-1}, e_i) aparece en el corpus $\mathcal{C}$, y $p(e_i \mid e_{i-2}, e_{i-1})$ es la probabilidad de que tras las palabras (e_{i-2}, e_{i-1}) aparezca la palabra e_i.

Hay que notar que

$$N(e_{i-2}, e_{i-1}) = \sum_{e' \in \mathcal{C}} N(e_{i-2}, e_{i-1}, e')$$

de modo que $p(\cdot)$ sigue una distribución de probabilidad, o sea:

$$\sum_{e' \in \mathcal{C}} p(e' \mid e_{i-2}, e_{i-1}) = 1.0$$

2.3.2 Suavizado del modelo

Los n-gramas pueden asignar probabilidades distintas de cero a frases no vistas con anterioridad (o sea, que no pertenezcan al corpus de entrenamiento). La única forma de poder obtener una probabilidad igual a cero sería si la frase contuviese algún n-grama que no haya sido visto anteriormente. En estos casos lo que se hace es aplicar suavizado.

Veamos como funciona el suavizado con un ejemplo. Supongamos que queremos calcular la probabilidad del trigrama *"x y z"*. Si *"z"* nunca sigue a *"x y"* en nuestro corpus, nos podríamos preguntar si al menos *"z"* siguió a *"y"*. Si así fue, probablemente el trigrama *"x y z"* no sea del todo malo. Si no, podríamos incluso preguntarnos si *"z"* es una palabra común o no. Si incluso no es una palabra común, entonces *"x y z"* podría tomar una probabilidad baja. Entonces en vez de utilizar la ecuación (2.3) para estimar la probabilidad del trigrama podríamos utilizar esta otra:

$$p(z \mid x\,y) = \lambda_1 \cdot \frac{N(x\,y\,z)}{N(x\,y)} + \lambda_2 \cdot \frac{N(y\,z)}{N(z)} + \lambda_3 \cdot \frac{N(z)}{N} + \lambda_4$$

con $1 \leq \lambda_i \leq 1$, $\sum_i \lambda_i = 1$ y $N = \mid \mathcal{C} \mid$.

Sería apropiado usar diferentes coeficientes λ_i de suavizado en diferentes situaciones, de modo que estos coeficientes deberán también ser estimados utilizando un conjunto/corpus de validación. Esta forma de suavizado es la más

simple y se conoce cómo técnica de interpolación lineal, aunque existen infinidad de técnicas de suavizado y estimación de parámetros para modelos de lenguaje. Para un estudio detallado sobre las técnicas de suavizado y de estimación de modelos de n-gramas referimos al lector a [Manning and Schütze 01].

Por último comentar que en nuestro ejemplo, el hecho de que en el peor de los casos asignemos una probabilidad de λ_4, nunca se podrá dar el caso de que la probabilidad asociada a un n-grama sea cero (al fin y al cabo la gente dice cosas gramaticalmente incorrectas cuando habla).

Evidentemente los n-gramas no es todo lo que podríamos hacer, sino que podríamos desarrollar otro tipo de modelos de lenguaje que tengan en cuenta la estructura gramatical del lenguaje, o que se basen en otro tipo de conocimiento lingüístico en general. De todas formas, en la actualidad y a pesar de la gran cantidad de investigación destinada a desarrollar otros modelos de lenguaje, es raro encontrar alguna técnica que supere a los n-gramas. Otra ventaja a su favor es la relativa sencillez de implementación y la existencia de gran cantidad de programas de libre distribución disponibles, por ejemplo el SLM [Rosenfeld and Clarkson 97] y el SRILM [Stolcke 97].

2.3.3 Evaluación de modelos

Lo que a priori sí parece interesante es poder saber si un modelo funciona mejor que otro para tener un criterio a la hora de elegir un modelo de lenguaje apropiado a la tarea que estemos abordando en cierto momento.

Entonces, ¿cómo podemos saber si un modelo funciona mejor que otro? Una forma sería coger un trozo de texto previamente no visto (no contenido en el conjunto de entrenamiento), es decir un conjunto de TEST y preguntar a cada modelo cuál es la probabilidad que le asignaría de acuerdo a la historia generativa y unos valores en particular de los parámetros. Esto se puede escribir de forma simbólica del siguiente modo:

$$Pr(\text{modelo} \mid \text{TEST})$$

Usando la regla de Bayes:

$$Pr(\text{modelo} \mid \text{TEST}) = Pr(\text{modelo}) * Pr(\text{TEST} \mid \text{modelo})/Pr(\text{TEST})$$

Supongamos que $Pr(\text{modelo})$ es la misma para todos los modelos. Esto es, sin mirar al conjunto de TEST, no tenemos idea de si 0.95 es un número mejor que 0.07 dentro de uno o distintos modelos. Entonces, el mejor modelo será el que maximice $Pr(\text{TEST} \mid \text{modelo})$. Afortunadamente, $Pr(\text{TEST} \mid \text{modelo})$ es algo fácil de calcular. Es justo lo mismo que $Pr(\mathbf{e})$, cuando $\mathbf{e} = \text{TEST}$. Nótese que en este caso $\mathbf{e}$ hace referencia a un conjunto de test el cual podrá ser una única frase o un conjunto de frases indistintamente.

Ahora, cualquiera podría hacer un programa que produzca valores para $Pr(\mathbf{e})$ y compararlo con cualquier otro programa. Esto en cierto modo es como hacer

apuestas, un modelo apostará cierta cantidad a todo tipo de frases, asignándoles una *puntuación* $Pr(\mathbf{e})$ mayor o menor. Cuando se le ofrece un conjunto de TEST el modelo apostará por ese TEST, de modo que cuanto más apueste por él mejor será el modelo.

En general un modelo de trigramas será mejor que uno de bigramas. Veamos esto con un ejemplo: un modelo de bigramas asignará una probabilidad relativamente alta a una frase como: *I hire men who is good pilots*, puesto que la frase contiene buenos pares de palabras. Siguiendo con el símil de las apuestas, un modelo de bigramas apostará una gran cantidad por esta frase. En cambio un modelo de trigramas no apostará tanto por ella debido a que el valor de $p(\textit{is} \mid \textit{men who})$ es muy bajo. Esto significa que el modelo de trigramas tendrá mayor cantidad para apostar por aquellas frases que tiendan a aparece en el conjunto de TEST, como por ejemplo en la frase *I hire men who are good pilots*.

Como hemos comentado anteriormente cualquier modelo que asigne una probabilidad de cero a un conjunto de TEST debe ser descartado. Como solución a ello hemos propuesto suavizar el modelo, y de hecho existen métodos para elegir los coeficientes de suavizado de modo que se optimice la $Pr(\text{TEST} \mid \text{modelo})$. Si queremos ser justos no deberemos tener en cuenta el conjunto de TEST para entrenar los parámetros del modelo, entonces para obtener esos coeficientes lo que se suele hacer es dividir el conjunto de ENTRENAMIENTO en dos subconjuntos (un conjunto de DESARROLLO y un conjunto de VALIDACIÓN) (ENTRENAMIENTO = DESARROLLO + VALIDACIÓN)). El primero (DESARROLLO) se utilizará para calcular las frecuencias/probabilidades de los n-gramas y el segundo (VALIDACIÓN) para establecer los valores de los coeficientes de suavizado de modo que se optimice la probabilidad $Pr(\text{VALIDACIÓN} \mid \text{modelo})$. Entonces ya podremos testear nuestros modelos con conjuntos de TEST no vistos previamente.

2.3.4 Perplejidad

Si el conjunto de TEST es muy grande, entonces un n-grama le asignará un valor de $Pr(\text{TEST})$ que será el producto de gran cantidad de números pequeños (todos ellos menores o igual que 1.0). Las probabilidades condicionales de algunos n-gramas pueden ser muy pequeñas con lo que $Pr(\text{TEST})$ será minúsculo. Tenemos por tanto dos problemas, uno es que al trabajar con computadores que tienen una precisión delimitada tendremos problemas de desbordamiento; y el segundo es como comparar dos conjuntos de TEST de tamaños diferente, puesto que el mayor de ellos en principio tendrá una probabilidad mucho más pequeña que el otro.

El primer problema lo solucionamos utilizando el logaritmo de la probabilidad en vez de la probabilidad propiamente dicha. Y el segundo teniendo en cuenta que una forma convencional de comparar modelos es normalizar con respecto al tamaño del conjunto de TEST considerado.

Por lo tanto, la *perplejidad* o entropía cruzada de un conjunto de TEST para

un modelo de lenguaje se define como:

$$2^{-\left(\frac{\sum_{\mathbf{e}\in \mathrm{TEST}} \log(Pr(\mathbf{e}))}{N}\right)}$$

donde N es el número de palabras que contiene el conjunto de TEST. Por tanto dividir por N ayuda a normalizar los números, de modo que un modelo tendrá aproximadamente la misma perplejidad sin que el tamaño del conjunto de TEST influya en ello.

La perplejidad es inversamente proporcional a la $Pr(\mathbf{e})$, de modo que un buen modelo de lenguaje tendrá un perplejidad relativamente pequeña y del mismo modo un valor de $Pr(\mathbf{e})$ relativamente alto. A menor perplejidad mejor modelo de lenguaje.

2.3.5 Modelos de lenguaje basados en categorías

Idealmente nos gustaría considerar una gran longitud en la historia de un modelo de n-gramas puesto que podrían existir dependencias a largo plazo que podríamos tener en cuenta. El problema es que n-gramas con historias muy largas aparecen con muy poca frecuencia, lo que llevaría a obtener una estimación de sus parámetros bastante mala.

En vez de esto lo que se suele hacer, para considerar historias más largas (en general de 3), es utilizar modelos de lenguaje basados en categorías/clases [Brown et al. 92], dado que a priori el número de clases será bastante reducido y por tanto la variabilidad de los n-gramas será sensiblemente inferior. En este caso las palabras se categorizan/agrupan en clases (bien sintácticas como en [Jelinek et al. 90, Kneser and Ney 93a, Jardino and Adda 93, Kneser and Ney 93b]), o semánticas como en [Brown et al. 92]).

A la clase, del tipo que sea, de una palabra $e \in \mathcal{E}$ la denotaremos con $\mathcal{C}(e)$. Por tanto un modelo de n-gramas basado en clases se define como:

$$Pr(e_1^I) = \prod_{i=1}^{I+1} p(e_i \mid \mathcal{C}(e_i)) \cdot p(\mathcal{C}(e_i) \mid \mathcal{C}(e_{i-n+1}), \ldots, \mathcal{C}(e_{i-1})) \tag{2.4}$$

donde:

- $p(e_i \mid \mathcal{C}(e_i))$ denota la probabilidad de pertenencia de la palabra e_i en la clase $\mathcal{C}(e_i)$, y
- $p(\mathcal{C}(e_i) \mid \mathcal{C}(e_{i-n+1}), \ldots, \mathcal{C}(e_{i-1}))$ es la probabilidad de la clase a la que pertenece la palabra e_i dada la historia de las clases a las que pertenecen las $n-1$ palabras anteriores a ella en la historia de la frase.

Normalmente las clases se construyen de forma automática [Brown et al. 92, Och 99]. En esta tesis realizaremos experimentos con el primer tipo de modelo de lenguaje, es decir modelos de lenguaje basados en palabras.

2.4 Modelos de traducción

Un modelo de traducción p de un lenguaje fuente ($\mathcal{E}$) a otro lenguaje destino ($\mathcal{F}$) con una serie de parámetros θ, es una fórmula para calcular una distribución de probabilidad, o verosimilitud, $p_\theta(\mathbf{f} \mid \mathbf{e})$, para cualquier frase $\mathbf{e}$ de $\mathcal{E}$ y cualquier frase $\mathbf{f}$ de $\mathcal{F}$. Estas probabilidades satisfacen las siguientes propiedades o restricciones:

$$\begin{aligned} p_\theta(\mathbf{f} \mid \mathbf{e}) \geq 0, \quad p_\theta(fallo \mid \mathbf{e}) \geq 0, \\ p_\theta(fallo \mid \mathbf{e}) + \sum_{\mathbf{f}} p_\theta(\mathbf{f} \mid \mathbf{e}) = 1 \end{aligned} \tag{2.5}$$

donde $fallo$ es un símbolo especial que no pertenece al vocabulario destino. $p_\theta(\mathbf{f} \mid \mathbf{e})$ se interpreta como la probabilidad de que un traductor genere la frase $\mathbf{f}$ dada la frase $\mathbf{e}$, y $p_\theta(fallo \mid \mathbf{e})$ como la probabilidad de no producir traducción alguna dada la frase $\mathbf{e}$. Se dice que un modelo es deficiente si $p_\theta(fallo \mid \mathbf{e})$ es mayor que 0 para alguna frase $\mathbf{e}$.

En este caso, como en todo problema de estimación/aprendizaje paramétrico, se pretende encontrar los parámetros que maximicen cierta función objetivo a partir de un conjunto de muestras de entrenamiento. En el caso de traducción automática estadística esta función objetivo es la función *logaritmo de la verosimilitud*, la cual para un conjunto/corpus de traducciones dado $\{(\mathbf{f}_s, \mathbf{e}_s) : s = 1, \ldots, S\}$ se define como:

$$\mathcal{L}(p_\theta) = S^{-1} \sum_{s=1}^{S} \log p_\theta(\mathbf{f}_s \mid \mathbf{e}_s) = \sum_{\mathbf{f},\mathbf{e}} C(\mathbf{f}, \mathbf{e}) \log p_\theta(\mathbf{f} \mid \mathbf{e}) \tag{2.6}$$

donde $C(\mathbf{f}, \mathbf{e})$ es la distribución empírica de la muestra, y es igual a $1/S$ veces el número de veces (normalmente 0 o 1) que el par $(\mathbf{f}, \mathbf{e})$ ocurre en la muestra. En definitiva se trata de obtener los valores de los parámetros θ que maximicen el logaritmo de la verosimilitud para un conjunto de muestras de entrenamiento. Notar que la fórmula (2.6) es exactamente la misma que la ecuación (1.3), sólo que en esta última se asume que $C(\mathbf{f}, \mathbf{e}) = 1$ para todo par $(\mathbf{f}, \mathbf{e})$.

2.4.1 La traducción como proceso de reescritura

Desde un punto de vista generativo, un modelo de traducción $Pr(\mathbf{f}|\mathbf{e})$ se puede ver como un modelo que genera una frase $\mathbf{f}$ a partir de una frase $\mathbf{e}$ siguiendo una historia generativa. Esta historia generativa la podemos como ver el proceso de distorsión que sufre la frase $\mathbf{e}$ cuando se transmite por un canal ruidoso para convertirse en $\mathbf{f}$, en que cada una de las palabras que conforman la frase $\mathbf{e}$ se conviertan en las palabras que conforman la frase $\mathbf{f}$, en un orden que, a priori, no tiene por que ser el mismo.

Veamos esta historia generativa en más detalle. Desde un punto de vista lingüístico, está claro que no podemos pretender que una frase $\mathbf{e}$ se convierta en

otra **f** simplemente remplazando las palabras una a una. De modo que deberemos permitir que algunas palabras en **e** puedan generar más de una en **f** o por el contrario que no generen palabra alguna. Esta historia generativa la podemos ver como un proceso de reescritura de cadenas [Knight 99b]. Veamos todo esto con un ejemplo: Primero empezaríamos con una frase como:

John did not kick the black cat

Entonces, decidimos cuántas palabras genera cada una de las palabras que la forman (llamémosle a este número ϕ)[5], de modo que si $\phi = 1$ simplemente copiamos la palabra en cuestión; si $\phi = 2$ la duplicamos; etcétera. Si $\phi = 0$ la eliminamos. En nuestro caso si decidimos que $\phi_{kick} = 3$, $\phi_{the} = 2$, $\phi_{did} = 0$ y $\phi_{John} = \phi_{black} = \phi_{cat} = 1$ nos quedaría:

John not kick kick kick the the black cat

A continuación remplazamos cada palabra por una de sus posibles traducciones.

John no dio una patada a la negra gata

Finalmente permutamos las palabras para obtener:

John no dio una patada a la gata negra

Evidentemente esta es una de las posibles (entre muchas) formas que podemos convertir la frase *John did not kick the black cat* en *John no dio una patada a la gata negra*.

2.4.2 Alineamientos entre frases

La historia generativa comentada en el punto anterior se puede resumir mediante el concepto de alineamiento. Un alineamiento **a** se define como una relación estructural entre un par de frases. En principio un alineamiento puede ser muy complicado, pues puede incluir efectos como cambios de orden entre las palabras, borrados, inserciones, y relaciones entre grupos de palabras. En general un alineamiento $\mathcal{A}$, entre una frase $\mathbf{f} = f_1^J = f_1, \ldots, f_j, \ldots, f_J$ en un lenguaje origen y una frase $\mathbf{e} = e_1^I = e_1, \ldots, e_i, \ldots, e_I$ en un lenguaje destino, se define como un subconjunto del producto cartesiano entre las posiciones de ambas frases, es decir:

$$\mathcal{A} \subseteq \{(j, i) : j = 1, \ldots, J; i = 1, \ldots, I\}$$

Modelar alineamientos de este tipo entre dos lenguajes origen y destino es difícil, de modo que los modelos de alineamiento que podemos encontrar en la bibliografía establecen una serie de restricciones para poder atacar el problema de forma factible.

De hecho los modelos de alineamiento con los que trabajaremos en esta tesis se restringen en la medida que cada palabra de una frase de entrada se alinea exactamente con una y sólo una palabra de la frase destino. Estos modelos de alineamiento son muy similares a los Modelos Ocultos de Markov (MOM) ampliamente utilizados en reconocimiento automático del habla. De este modo

[5] Formalmente al número de palabras que genera una palabra e_i le llamaremos *fertilidad*.

un alineamiento $\mathbf{a} = a_1^J$ es un mapeo que asigna a cada posición origen j una posición destino i, o sea $j \rightarrow i = a_j$. El alineamiento $a_1^J = a_1, \ldots, a_j, \ldots, a_J$ puede contener valores $a_j = 0$, de modo que algunas palabras origen se pueden alinear con la palabra e_0 de la frase destino. Esta palabra ficticia e_0 se utiliza conceptualmente para permitir la aparición de palabras espurias en el proceso generativo, o sea para permitir que haya palabras en la estructura origen que no se alineen con ninguna palabra destino. A la palabra e_0 se le llama palabra nula ($NULL$) o palabra vacía.

2.4.3 Modelos estadísticos de alineamiento

Como ya hemos comentado anteriormente, en la traducción automática estadística intentamos modelar la probabilidad de traducción $Pr(\mathbf{f}|\mathbf{e})$. Esta probabilidad describe la relación entre un par de frases origen-destino. En un modelo estadístico de alineamiento $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$, se introduce el alineamiento $\mathbf{a} = a_1^J$ como variable oculta, de modo que la relación entre un modelo de traducción y un modelo de alineamiento viene dada por:

$$Pr(\mathbf{f}|\mathbf{e}) = \sum_{\mathbf{a}} Pr(\mathbf{f}, \mathbf{a}|\mathbf{e}) \tag{2.7}$$

Evidentemente el arte del modelado estadístico consiste en desarrollar modelos de traducción que capturen las propiedades relevantes del problema considerado dentro de un dominio específico. Por lo tanto, un modelo estadístico de alineamiento deberá describir convenientemente la relación entre un par de frases origen-destino.

Una vez más, un modelo de alineamiento dependerá de una serie de parámetros θ, los cuales deberán ser estimados maximizando la verosimilitud para un conjunto/corpus de pares de frases entrenamiento $\{(\mathbf{f}_s, \mathbf{e}_s) : s = 1, \ldots, S\}$, por lo tanto el conjunto óptimo de parámetros vendrá dado por la expresión:

$$\hat{\theta} = \arg\max_{\theta} \left\{ \prod_{s=1}^{S} \left[\sum_{\mathbf{a}} p_\theta(\mathbf{f}_s, \mathbf{a} \mid \mathbf{e}_s) \right] \right\} \tag{2.8}$$

Para realizar esta maximización, normalmente se utiliza el algoritmo EM (Expectation-Maximization)[Dempster et al. 77] o alguna aproximación a él.

Alineamiento por Viterbi

La ecuación (2.7) nos dice que un modelo de traducción recoge todas las posibles formas que tenemos de convertir una frase $\mathbf{e}$ en otra $\mathbf{f}$, es decir, es la suma para todo posible alineamiento de la probabilidad que el modelo estadístico de alineamiento asigna a cada alineamiento posible. Evidentemente habrá alineamientos más probables que otros, por ejemplo, el alineamiento de la figura 2.1 parece que podría tener una probabilidad bastante alta. De entre todos esos

posibles alineamientos habrá uno que será el más probable (de acuerdo con los parámetros del modelo) y se puede obtener como:

$$\hat{\mathbf{a}} = \arg\max_{\mathbf{a}} p_{\hat{\theta}}(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) \tag{2.9}$$

A este alineamiento $\hat{\mathbf{a}} = \hat{a}_1^J$ más probable se le conoce cómo alineamiento por *Viterbi*.

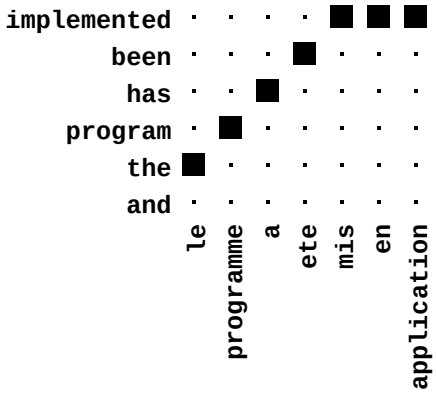


Figura 2.1: Un ejemplo de alineamiento entre inglés y francés.

2.4.4 Evaluación de modelos

Al igual que hemos comentado para el caso de modelos de lenguajes también podemos evaluar/comparar la calidad de varios modelos de traducción, y para ello definimos la *perplejidad* o entropía cruzada de un conjunto de TEST para un modelo de traducción del siguiente modo:

$$2^{-\left(\frac{\sum_{\mathbf{e},\mathbf{f}\in \mathrm{TEST}} \log p(\mathbf{f}|\mathbf{e})}{N}\right)}$$

donde N es igual a número de palabras que contiene el conjunto de TEST de entrada.

2.5 El problema de la búsqueda y la complejidad de la tarea de la traducción automática

La complejidad de la tarea de la traducción automática estadística está sujeta a diversas consideraciones [Germann et al. 01]: si el orden de las palabras de

la frase destino se restringe al mismo que el de las de la frase origen, entonces la complejidad de la búsqueda es lineal. Si lo único que se permiten son rotaciones entre los nodos de un árbol binario, entonces la complejidad del algoritmo es polinomial. Por último, si las palabras pueden reordenarse arbitrariamente, entonces la complejidad es exponencial lo que convierte el problema de la búsqueda en un problema NP-completo [Knight 99a]. Este último punto de vista es el que normalmente se sigue en la mayoría de los sistemas estadísticos de traducción automática existentes y el mismo que utilizaremos en esta tesis para abordar el problema de la búsqueda (o decodificación), lo cual en principio hace que la tarea de la traducción automática sea un problema inabordable.

2.5.1 Errores en el proceso de búsqueda. Modelo de error

Supongamos que $E = \{\hat{\mathbf{e}}_i, 1 \leq i \leq n\}$ es el conjunto de todas las posibles traducciones correctas de la frase origen $\mathbf{f}$. Si un decodificador devuelve $\mathbf{e}$ como hipótesis de la traducción de $\mathbf{f}$, entonces estaremos ante dos posibles situaciones:

- Que $\mathbf{e} \in E$, en este caso decimos que se ha producido un *acierto* en la traducción.
- Que $\mathbf{e} \notin E$, con lo que se habrá producido un *error* en la traducción.

Siguiendo esta filosofía, si se desea que un decodificador no produzca errores de traducción, entonces todo el espacio de búsqueda debe ser explorado, lo que implica enfrentarse a una complejidad no polinomial como ya hemos comentado. Para reducir esta complejidad la mayoría de los algoritmos de decodificación existentes (véase por ejemplo [Wang and Waibel 97]) restringen el espacio de búsqueda, de manera que sólo se examinen grandes subconjuntos del mismo. Lógicamente, tal medida supone correr el riesgo de no poder garantizar alcanzar la traducción óptima, es decir la mejor de todas las posibles de acuerdo a los modelos utilizados.

De cualquier forma, en el caso que nos compete, la bondad de una hipótesis $\mathbf{e}$ cómo traducción de una frase $\mathbf{f}$ vendrá determinada por la bondad de los modelos de lenguaje y traducción que estemos utilizando. Por tanto estamos en disposición de afirmar que, incluso en el hipotético caso de disponer de un mecanismo capaz de obtener una traducción óptima, no podemos asegurar que esa traducción sea una buena traducción desde el punto de vista de un experto.

Por este motivo, en la práctica, y con la esperanza de poder evaluar un sistema de traducción de forma automática, se utilizan un conjunto frases de referencia E_{ref}, que a juicio de los expertos sean buenas traducciones de una frase dada. Estas traducciones tendrán a su vez asociados los valores de la probabilidad que les asignen los modelos utilizados, y por lo tanto, se puede establecer un ordenamiento de dicho conjunto de acuerdo a dicha probabilidad. De este modo, si suponemos que: $\mathbf{e}_m$ es la traducción de referencia menos probable del conjunto E_{ref}; $\mathbf{e}_M$ es la traducción de referencia más probable del conjunto E_{ref}; $\mathbf{e}$ es

la hipótesis que devuelve un decodificador como traducción de la frase origen $\mathbf{f}$, entonces estaremos ante tres posibles situaciones:

- Que $\mathbf{e} \in E_{ref}$, en este caso decimos que se ha producido un *acierto* en la traducción.
- Que $\mathbf{e} \notin E_{ref}$ y $P(\mathbf{e} \mid \mathbf{f}) < P(\mathbf{e}_m \mid \mathbf{f})$. En este caso decimos que se habrá producido un *error de búsqueda*. Es decir, el decodificador no ha sido capaz de encontrar ninguna de las traducciones pertenecientes al conjunto E_{ref} debido a la reducción del espacio de búsqueda realizado.
- Que $\mathbf{e} \notin E_{ref}$ y $P(\mathbf{e} \mid \mathbf{f}) > P(\mathbf{e}_M \mid \mathbf{f})$. En este caso decimos que se ha producido un *error del modelo*, lo que significa que según el modelo hay otra hipótesis ($\mathbf{e}$ en este caso) con una *puntuación* mejor que cualquiera de las pertenecientes al conjunto E_{ref}. Este tipo de error no es, por tanto, atribuible al decodificador.

Esta última clasificación de los errores de traducción o *Modelo de Error* es mucho más apropiada en nuestro caso a la hora de evaluar la eficacia de los algoritmos de búsqueda que se exponen en esta tesis. Desafortunadamente en la mayoría de los corpus de experimentación que utilizaremos sólo vamos a disponer de una frase de referencia, con lo que los resultados y conclusiones que se saquen a este efecto estarán condicionados siempre a esta grave restricción.

2.6 Pre y postprocesos

Como hemos comentado en el capítulo de introducción dentro de la arquitectura de un sistema de TAE (ver figura 1.4) existe una fase de preprocesado y otra de postprocesado.

Normalmente en la TAE nos encontramos con el problema de falta de datos de entrenamiento. Hoy en día, y gracias al ámbito plurilingüe en que nos desarrollamos, a la facilidad de almacenar datos masivamente y a la gran difusión de información que Internet ha provocado, este problema se ve parcialmente aliviado.

A pesar de ello gran cantidad de palabras y de construcciones sintácticas aparecen una sola vez en la mayoría de los corpus de entrenamiento existentes[6]. Por este motivo, es prácticamente imposible construir modelos estadísticos de lenguaje y traducción robustos. Además, siempre es necesario realizar asunciones en los modelos para hacer practicable la estimación de sus parámetros y permitir obtener eficientes algoritmos de búsqueda. Por lo tanto, debido a estas simplificaciones, es imposible capturar dentro de los modelos ciertos fenómenos bastante comunes en el lenguaje natural, lo cual da lugar a que, en gran cantidad de casos, el sistema de traducción obtenga traducciones erróneas.

[6] Algunas estadísticas hablan de que el 90% de las palabras encontradas en un corpus equivalen tan solo al 10% del tamaño del vocabulario empleado.

Una solución parcial a este problema pasa por realizar ciertas operaciones de preproceso (y el consiguiente postproceso) para reducir en la medida de lo posible la complejidad de la tarea a abordar. De hecho, en la mayoría de los sistemas de TAE existentes existe una fase de preproceso, la cual ayuda a obtener mejores resultados con un esfuerzo relativamente pequeño. Ejemplos típicos de preprocesos son: la categorización de ciertas palabras (o grupos de palabras) que potencialmente conllevan un vocabulario infinito, como son los nombres propios, los números, las siglas y las fechas; el reordenado de palabras, lo cual simplifica el proceso de traducción al permitir monotonicidad y reducir las dependencias a largo término entre componentes sintácticos a dependencias locales; e incluso el uso de lematizadores o analizadores morfosintácticos para reducir la variabilidad en lenguajes altamente declinados.

Como ya hemos comentado en la sección 1.4.2 la fase de preproceso se puede aplicar en la fase de aprendizaje/entrenamiento del sistema de traducción, lo que conllevaría aplicar el preproceso a los lenguajes origen y destino de la traducción. Del mismo modo también habría que aplicar el mismo preproceso a una frase de entrada a traducir. El postproceso solo habrá que aplicarlo cuando se aplique el preproceso a la entrada y será equivalente a realizar el proceso inverso al anterior.

Preproceso

Formalmente el preproceso consiste en la aplicación de sendas funciones a las frases de los lenguajes origen y destino de la traducción. Es decir, existirán un par de funciones $prep_{\mathbf{f}}(\mathcal{F}^*) : \mathcal{F'}^*$ y $prep_{\mathbf{e}}(\mathcal{E}^*) : \mathcal{E'}^*$ con la particularidad deseable de que cumplan que: $|\mathcal{F'}| < |\mathcal{F}|$ y $|\mathcal{E'}| < |\mathcal{E}|$, es decir que el tamaño de los vocabularios del corpus preprocesado sea menores con respecto a los tamaños de los vocabularios sin preprocesar. Por lo tanto la fase de preproceso consistirá en la aplicación de estas funciones a toda frase del corpus de entrenamiento $\mathcal{C}$:

$$\forall (f_1^J, e_1^I) \in \mathcal{C} : prep_{\mathbf{f}}(f_1^J) \rightarrow {f'}_1^{J'},\ prep_{\mathbf{e}}(e_1^I) \rightarrow {e'}_1^{I'}$$

para obtener un corpus de entrenamiento distinto $\mathcal{C'}$ que permita obtener una perplejidad menor (o equivalentemente una mayor verosimilitud) para los modelos de traducción $Pr({f'}_1^{J'} \mid {e'}_1^{I'})$ y de lenguaje $Pr({e'}_1^{I'})$.

Postproceso

En cuanto al postproceso hay que comentar que solamente es necesario aplicarlo en la salida, es decir a la hora de traducir una frase de entrada f_1^J habrá que seguir los siguientes pasos:

1. Convertir f_1^J en ${f'}_1^{J'}$ utilizando la función $prep_{\mathbf{f}}()$
2. Traducir ${f'}_1^{J'}$ en la correspondiente ${e'}_1^{I'}$

3. Convertir $e'^{I'}_1$ en e^I_1 utilizando la función de postproceso $prep^{-1}_{\mathbf{e}}(\mathcal{E}'^*) : \mathcal{E}^*$.

En principio la función $prep^{-1}_{\mathbf{e}}()$ no tendrá porqué ser exactamente la inversa de $prep_{\mathbf{e}}()$, pero si deberá mantener el significado de la frase. Nótese también que en un esquema de este tipo no es necesario una función de postproceso $prep^{-1}_{\mathbf{f}}()$ para las frases de entrada.

2.7 Corpus de experimentación

A continuación describiremos las características más relevantes de los corpus/tareas que se han utilizado en el desarrollo y experimentación presentados en esta tesis.

2.7.1 La tarea del turista (EuTrans-I)

La tarea del turista o EuTrans-I[7], es una subtarea de la tarea del viajante ("Traveler Task") [Vidal 97, Amengual et al. 00] consistente en un corpus de traducción del castellano al inglés, generado semi-automáticamente en base a libros de bolsillo, guías rápidas y manuales del trotamundos. El dominio de esta tarea consiste en situaciones típicas de comunicación persona a persona en un mostrador de un hotel. Para esta tarea existe un procedimiento de categorización automática, mediante la cual se remplazan números, fechas y nombres propios por etiquetas asociadas a dichas categorías. Un resumen de las estadísticas de este corpus lo podemos ver en la tabla 2.2.

Dada la naturaleza semiautomática de la tarea y del reducido vocabulario que posee esta tarea se considera una tarea sencilla para abordar el problema de la traducción. En algunos desarrollos presentados en esta tesis se utilizará esta tarea para sintonizar algunos parámetros, dada su sencillez y fácil manejo.

2.7.2 Verbmobil

La tarea Verbmobil [Wahlster 00] es una tarea de traducción de habla del alemán al inglés en el dominio de reservas de hotel, planificación de viajes y planificación de citas. Las frases bilingües usadas en el entrenamiento son transcripciones correctas de diálogos hablados reales, de modo que incluyen difluencias, titubeos a la hora de hablar, falsos inicios de frases y frases gramaticalmente erróneas. Por lo tanto se considera una tarea muy difícil de abordar.

El objeto del proyecto Verbmobil fue la traducción de diálogos dentro del dominio de planificación de viajes y citas. En este proyecto se recopilaron cierta cantidad de diálogos reales, los cuales fueron manualmente transcritos y traducidos por algunos de los participantes en el proyecto. Debido a la gran variedad

[7] El nombre de EuTrans-I viene de que fue la tarea definida para evaluar un prototipo de traducción desarrollado en el marco del un proyecto europeo llamado EuTrans

		Castellano	Inglés
Entrenamiento	Frases	10 000	
	Palabras	97 131	99 292
	Palabras*	78 783	85 797
	Vocabulario	686	513
Test	Frases	2 996	
	Palabras	35 023	35 590
	Perplejidad (Bigramas)	–	5.2
	Perplejidad (Trigramas)	–	3.6

Tabla 2.2: Estadísticas del corpus EuTrans-I. (Palabras* sin considerar los signos de puntuación).

		Alemán	Inglés
Entrenamiento	Frases	58 073	
	Palabras	519 523	549 921
	Vocabulario	7 940	4 673
Test	Frases	251	
	Palabras	2 628	2 871
	Perplejidad (Trigramas)	–	30.5

Tabla 2.3: Estadísticas del corpus Verbmobil.

de traductores involucrados en el proceso de traducción dotó a este corpus de una gran variedad en el tipo/calidad de las traducciones.

Las estadísticas de este corpus las podemos ver en la tabla 2.3.

2.7.3 El corpus Hansards

La tarea Hansards es un corpus de traducción del francés al inglés, y posee un vocabulario muy grande de alrededor de 100 000 palabras para el francés y 80 000 para el inglés.

El corpus Hansards consiste en las actas del parlamento canadiense, las cuales se mantiene (por ley) en ambos idiomas debido al bilingüismo existente en este país. Existen alrededor de 3 millones de pares de frases, las cuales han sido publicadas por el LDC (Linguistic Data Consortium)[8] para fines de investigación y desarrollo. Nosotros utilizaremos un subcorpus de este, en el que solo se consideran frases cuya longitud máxima no exceda las 30 palabras. Las estadísticas de este subcorpus las podemos ver en la tabla 2.4.

Esta tarea también se considera una tarea difícil de abordar, dado el gran

[8] http://www.ldc.upenn.edu/

		Francés	Inglés
Entrenamiento	Frases	128 000	
	Palabras	1 931 405	2 120 212
	Vocabulario	37 542	29 414
Test	Frases	500	
	Palabras	4 000	3 896
	Perplejidad (Trigramas)	–	179.8

Tabla 2.4: Estadísticas del corpus HANSARDS.

tamaño del vocabulario que posee. A pesar de ello no resulta tan compleja como la tarea VERBMOBIL dada la gran cantidad de frases disponibles.

2.8 Evaluación de la traducción

Hasta la fecha no existe un criterio estándar de cómo evaluar un sistema de traducción automática en general. Idealmente la mejor forma de evaluar la calidad de un traductor automático pasa por la evaluación humana, es decir que la salida del sistema sea analizada detenidamente por un grupo de expertos lingüistas y que de alguna manera determinan la corrección (a distintos niveles, léxico, sintáctico, semántico) de las traducciones obtenidas. A este tipo de evaluación se le conoce como evaluación subjetiva.

La evaluación subjetiva tiene por contra otros inconvenientes:

- Por un lado se perdería todo el atractivo que caracteriza a un sistema de traducción automática, que es precisamente el realizar traducciones de forma completamente automática. Si por ejemplo, una vez desarrolladas las herramientas necesarias, un sistema de TAE necesita de unas pocas semanas para tener un sistema operativo para una tarea nueva, este tipo de evaluación requeriría posiblemente meses de trabajo.

- Por otro lado no es fácil realizar una comparativa entre distintos sistemas debido a que es difícil garantizar que la evaluación se haya realizado por el mismo grupo de expertos, en el mismo momento y bajo las mismas circunstancias. Una posible solución a este problema conllevaría el uso de bases de datos y herramientas comunes cómo se propone en [Jones and Rusk 00, Nießen et al. 00, Vogel et al. 00].

Por todo ello se ha tendido a intentar evaluar los sistemas de TA de forma también automática. De hecho en las pocas evaluaciones oficiales que hasta la fecha se han realizado por la organización ARPA-NIST, en todas ellas se han utilizado métodos de evaluación automáticos, a pesar de la crítica suscitada por determinados sectores lingüísticos.

Por todo ello, en esta tesis utilizaremos una serie de métricas popularmente utilizadas dentro del área de la TA. A continuación exponemos las métricas más utilizadas en la bibliografía:

- WER (*word error rate*):
  Esta medida ha sido heredada de de la evaluación de sistemas de reconocimiento del habla dado que en este campo ha sido aceptada como estándar de evaluación. El WER consiste en el cálculo del mínimo número de operaciones de edición (sustituciones, borrados e inserciones) necesarias para convertir una frase generada por un traductor en otra considerada como traducción de referencia de la frase de entrada. Este mínimo se calcula mediante un algoritmo de programación dinámica, normalmente conocido como la distancia de Levenstein [Levenstein 65].

- PER (*position-independent word error rate*):
  Un problema del WER es que requiere que el orden de las palabras sea perfecto con respecto a la frase de referencia. Esto en particular es problemático para idiomas en los que el orden de las palabras puede ser bastante diferente como es el caso del alemán (y por lo tanto para la tarea VERBMOBIL), es decir, que a pesar de que el orden de las frases en la frase generada y la frase de referencia sea muy distinto, ambas podrían ser traducciones válidas de una frase de entrada. Este efecto no puede ser captado por el WER con lo que también utilizaremos la medida PER la cual hace exactamente lo mismo que el WER pero sin tener en cuenta el orden de las palabras, de modo que en el caso de palabras de la frase generada que no tengan contrapartida en la frase de referencia se contarán como errores de sustitución. En el caso en que las longitudes de ambas frases sean distintas, el resto de palabras darán lugar adicionalmente a errores de inserción o borrado. Evidentemente, el PER siempre será menor o igual que el WER.

- mWER (*multi-reference word error rate*):
  Para los casos en los que se disponga de un conjunto de frases de referencia válidas (o sea, varias posibles traducciones válidas para una misma frase de entrada) se aplicará esta medida, que pasa por calcular el WER para cada frase de referencia y quedarnos con el mejor. Para más detalles acerca de esta medida ver [Nießen et al. 00].

- SER (*sentence error rate*)
  Esta medida simplemente comparará íntegramente la frase generada con la frase de referencia, proporcionando un error siempre en cuanto ambas frases no sean exactamente iguales. Evidentemente esta medida es bastante pesimista y raramente se utiliza en la evaluación de sistemas de TA.

- BLEU *score*:
  Esta medida mide la precisión de unigramas, bigramas, trigramas y cuatro-

gramas con respecto a todo un conjunto de traducciones de referencia, penalizando aquellas frases muy cortas [Papineni et al. 01]. Por contra de las medidas anteriores el BLEU mide aciertos, con lo que a mayor *score* mejor traducción.

En los casos en que sea posible, y a falta de un grupo de expertos lingüistas, solamente utilizaremos una medida de evaluación subjetiva de evaluación para aquellas tareas en las que el dominio del lenguaje destino sea amplio (en nuestro caso, desafortunadamente, sólo para el castellano). La medida subjetiva que utilizaremos será el SSER (*subjetive sentence error rate*) y consiste en que cada frase generada será examinada por un humano y juzgará su corrección de acuerdo a una escala que varía entre 0.0 y 1.0 [Nießen et al. 00], de modo que un valor de 0.0 indicará que la traducción es sintáctica y semánticamente correcta, un valor de 0.5 indicará que la traducción es sintácticamente incorrecta pero semánticamente correcta, y un valor de 1.0 indicará que la traducción es semánticamente incorrecta.

Por último cabe comentar que los criterios de evaluación automáticos (a saber WER, PER, mWER y BLEU) en gran cantidad de casos están correlacionados con evaluaciones subjetivas, de hecho las medidas mWER y BLEU correlacionan especialmente bien con evaluaciones subjetivas.

2.9 Resumen

En este capítulo hemos expuesto con cierto nivel de detalle los componentes de un sistema estadístico de traducción automática, así como la forma de evaluarlo y los corpus de experimentación que utilizaremos en el resto de la tesis.

II

Modelos de traducción

CAPÍTULO 3

Modelos estadísticos de alineamiento

EN este capítulo describiremos con cierto nivel de detalle la familia de modelos de traducción de IBM propuestos en [Brown et al. 93], en los cuales se basarán los algoritmos de búsqueda que expondremos en esta tesis. También comentaremos brevemente otros modelos de traducción también ampliamente utilizados. En [Knight 99b] podemos encontrar una descripción más informal, aunque mucho más intuitiva, de algunos de los modelos de IBM.

3.1 Introducción

Como hemos comentado en la sección 2.4.3, la relación entre un modelo de traducción y un modelo de alineamiento viene dado por la expresión:

$$Pr(\mathbf{f} \mid \mathbf{e}) = \sum_{\mathbf{a}} Pr(\mathbf{f}, \mathbf{a} \mid \mathbf{e}), \tag{3.1}$$

de modo que la relación estructural entre un par de frases $(\mathbf{f}, \mathbf{e})$ viene dada por un alineamiento $\mathbf{a}$, que no es más que una forma de resumir el proceso generativo por el cual una frase $\mathbf{e}$ dio lugar a otra $\mathbf{f}$.

Un modelo de alineamiento $Pr(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$ se puede descomponer, sin pérdida de generalidad [Brown et al. 93], mediante la expresión:

$$Pr(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) = Pr(J \mid \mathbf{e}) \cdot \prod_{j=1}^{J} Pr(a_j \mid f_1^{j-1}, a_1^{j-1}, \mathbf{e}) \cdot Pr(f_j \mid f_1^{j-1}, a_1^{j}, \mathbf{e}) \tag{3.2}$$

en la que se obtienen tres probabilidades diferentes:

- Una probabilidad de longitudes: $Pr(J \mid \mathbf{e})$,
- Una probabilidad de alineamiento entre palabras: $Pr(a_j \mid f_1^{j-1}, a_1^{j-1}, \mathbf{e})$,
- Y una probabilidad de traducción entre palabras: $Pr(f_j \mid f_1^{j-1}, a_1^j, \mathbf{e})$.

De forma general, y viendo a un modelo de alineamiento como modelo generativo, estas tres probabilidades se pueden interpretar como: Primero se elige la longitud J de la frase $\mathbf{f}$ a generar (la cual dependerá de la frase $\mathbf{e}$) con una probabilidad $Pr(J \mid \mathbf{e})$; una vez conocida la longitud de la frase a generar, para cada palabra que forme parte de $\mathbf{f}$ primero se elegirá la posición a_j de $\mathbf{e}$ con la que se va a alinear, con una probabilidad $Pr(a_j \mid f_1^{j-1}, a_1^{j-1}, \mathbf{e})$ (dependiendo de la frase $\mathbf{e}$, las palabras previamente generadas f_1^{j-1} y sus respectivos alineamientos a_1^{j-1}), y por último se elegirá la palabra a ser generada con una probabilidad $Pr(f_j \mid f_1^{j-1}, a_1^j, \mathbf{e})$ (con las mismas dependencias que en el caso anterior además de a_j).

En la sección siguiente expondremos la familia de modelos estadísticos de alineamiento de IBM, en la que se harán asunciones específicas a estas distribuciones de probabilidad. Ya podemos adelantar que la gran diferencia entre los distintos modelos de IBM (1-5)[1], radica en la forma de modelar la distribución de probabilidad de alineamiento ($Pr(a_j \mid f_1^{j-1}, a_1^{j-1}, \mathbf{e})$), pasando por ser una constante para el modelo 1, hasta un expresión bastante compleja, con dependencias de primer orden, para los modelos 4 y 5.

Como hemos comentado en la sección 2.4.3, un modelo de alineamiento $Pr(f_1^J, a_1^J \mid \mathbf{e})$ dependerá de una serie de parámetros θ, los cuales se estimarán maximizando la verosimilitud del modelo con respecto a un corpus de entrenamiento. Los parámetros θ, vendrán definidos por las asunciones/restricciones que se impongan a cada modelo.

3.2 Los modelos de alineamiento de IBM

En lo que sigue, para cada modelo en concreto veremos qué asunciones se hacen en el modelo, y por tanto qué parámetros habrá que estimar para cada uno de ellos, describiendo en cada caso el proceso generativo de obtener una frase $\mathbf{f} = f_1^J$ y el correspondiente alineamiento $\mathbf{a} = a_1^J$ a partir de otra frase $\mathbf{e} = e_1^I$.

3.2.1 Los Modelos 1 y 2

La fórmula general para estos modelos es:

$$p_\theta(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) = p_\theta(J \mid \mathbf{e}) \cdot p_\theta(\mathbf{a} \mid J, \mathbf{e}) \cdot p_\theta(\mathbf{f} \mid \mathbf{a}, J, \mathbf{e}) \tag{3.3}$$

[1] En lo sucesivo nos referiremos a dichos modelos como modelo 1, modelo 2,$\cdots$, modelo 5; o simplemente como $1, 2, \cdots 5$ cuando no sea necesaria la palabra modelo en el contexto.

Para el modelo 2 hacemos las siguientes asunciones:

$$p_\theta(J \mid \mathbf{e}) = \epsilon(J \mid I) \tag{3.4}$$

$$p_\theta(\mathbf{a} \mid J, \mathbf{e}) = \prod_{j=1}^{J} a(a_j \mid j, J, I) \tag{3.5}$$

$$p_\theta(\mathbf{f} \mid \mathbf{a}, J, \mathbf{e}) = \prod_{j=1}^{J} t(f_j \mid e_{a_j}) \tag{3.6}$$

Por tanto los parámetros para el modelo 2 serán

$$\theta = \{\epsilon(\cdot), t(\cdot), a(\cdot)\}$$

con:

$\epsilon(J \mid I)$	probabilidades de longitudes
$t(f \mid e)$	probabilidades de traducción
$a(i = a_j \mid j, J, I)$	probabilidades de alineamiento

para todo $f \in \mathcal{F}; e \in \{\mathcal{E} \cup e_0\}; I = 1, 2, \cdots$; y $J = 1, 2, \cdots$

Las ecuaciones (3.3)-(3.6) describen el siguiente proceso generativo:

1. Elegir la longitud J de $\mathbf{f}$ de acuerdo a la distribución de probabilidad $\epsilon(J \mid I)$.

2. Para cada $j = 1, 2, \cdots J$, elegir $a_j \in 0, 1, 2, \cdots I$ de acuerdo a la distribución de probabilidad $a(a_j \mid j, J, I)$.

3. Para cada cada $j = 1, 2, \cdots J$, elegir un palabra f_j, de acuerdo a la distribución de probabilidad $t(f_j \mid e_{a_j})$.

Debido a las asunciones de independencia (3.4)-(3.6), la suma para todos los alineamientos (3.1) tiene una forma exacta, es decir:

$$p_\theta(\mathbf{f} \mid \mathbf{e}) = \sum_{\mathbf{a}} p_\theta(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) \tag{3.7}$$

$$= \epsilon(J \mid I) \sum_{a_1=0}^{I} \cdots \sum_{a_J=0}^{I} \prod_{j=1}^{J} t(f_j \mid e_{a_j}) \cdot a(a_j \mid j, J, I) \tag{3.8}$$

$$= \epsilon(J \mid I) \prod_{j=1}^{J} \sum_{i=0}^{I} t(f_j \mid e_i) \cdot a(i \mid j, J, I) \tag{3.9}$$

El cálculo de la ecuación (3.9) se puede realizar muy eficientemente dado que sólo requiere del orden de $O(I \cdot J)$ operaciones aritméticas, pues no es necesario

realizar la suma sobre todos los posibles alineamientos (3.8), lo cual requeriría el orden de $O(I^J)$ operaciones.

Normalmente en la práctica en el modelo de alineamiento $a(i \mid j, J, I)$ se elimina la dependencia con el valor de J para evitar el sobre-entrenamiento debido a la falta de datos de entrenamiento, por lo tanto cuando describamos los algoritmos de búsqueda para el modelo 2 utilizaremos una aproximación al modelo de alineamiento tal y como se describe en [Brown et al. 93], es decir utilizaremos el modelo $a(i \mid j, I)$.

La función objetivo (2.6), que recordamos aquí por claridad,

$$\mathcal{L}(p_\theta) = S^{-1} \sum_{s=1}^{S} \log p_\theta(\mathbf{f}_s \mid \mathbf{e}_s) = \sum_{\mathbf{f},\mathbf{e}} C(\mathbf{f}, \mathbf{e}) \log p_\theta(\mathbf{f} \mid \mathbf{e})$$

es una función estrictamente cóncava en sus parámetros [Brown et al. 93]. De hecho a partir de ella y de la ecuación (3.9) se obtiene que,

$$\begin{aligned} \mathcal{L}(p_\theta) &= \sum_{\mathbf{f},\mathbf{e}} C(\mathbf{f}, \mathbf{e}) \sum_{j=1}^{J} \log \sum_{i=0}^{I} t(f_j \mid e_i) a(i \mid j, J, I) + \\ &\quad \sum_{\mathbf{f},\mathbf{e}} C(\mathbf{f}, \mathbf{e}) \log \epsilon(J \mid I) \end{aligned} \tag{3.10}$$

la cual es claramente cóncava en $\epsilon(J \mid I)$, $t(f \mid e)$ y $a(i \mid j, J, I)$, dado que el logaritmo de una suma es cóncava, y la suma de funciones cóncavas es cóncava. Dado que $\mathcal{L}$ es cóncava, existe un único máximo local. Es más, nosotros encontraremos este máximo utilizando el algoritmo EM, siempre y cuando aseguremos que ninguno de nuestros parámetros tiene un valor inicial de cero.

Las mismas consideraciones podemos hacer para el modelo 1, pues no es más que un caso especial del modelo 2 en el cual las probabilidades de alineamiento se distribuyen uniformemente: $a(i \mid j, J, I) = 1/(I+1)$ para todo i. De modo que los parámetros para el modelo 1 serán:

$$\theta = \{\epsilon(\cdot), t(\cdot)\} \ .$$

Del mismo modo que para el modelo 2, la suma para todos los alineamientos tiene una forma exacta:

$$p_\theta(\mathbf{f} \mid \mathbf{e}) = \sum_{\mathbf{a}} p_\theta(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) \tag{3.11}$$

$$= \frac{\epsilon(J \mid I)}{(I+1)^J} \prod_{j=1}^{J} \sum_{i=0}^{I} t(f_j \mid e_i) \tag{3.12}$$

Estos modelos no son deficientes, dado que $p_\theta(fallo \mid \mathbf{e}) = 0$, o lo que es lo mismo $\sum_{\mathbf{f}} p_\theta(\mathbf{f} \mid \mathbf{e}) = 1$.

3.2.2 Alineamiento por Viterbi

Para los modelos 1 y 2, podemos expresar de forma exacta el alineamiento por Viterbi ($V(\mathbf{f} \mid \mathbf{e}) = \hat{a}_1^J$)[2] entre un par de frase (f_1^J, e_1^I):

$$\hat{a}_1^J = \arg\max_{a_1^J} Pr(f_1^J, a_1^J \mid e_1^I) \tag{3.13}$$

$$= \arg\max_{a_1^J} \left\{ \prod_{j=1}^{J} t(f_j \mid e_{a_j}) \cdot a(a_j \mid j, J, I) \right\} \ , \tag{3.14}$$

$$\tag{3.15}$$

o lo que es lo mismo:

$$\hat{a}_j = \arg\max_i \{ t(f_j \mid e_i) \cdot a(i \mid j, J, I) \} \ . \tag{3.16}$$

De modo que la maximización sobre los $(I+1)^J$ alineamientos diferentes se descompone en J maximizaciones de $(I+1)$ probabilidades léxicas. Por lo tanto, el coste de calcular el alineamiento por Viterbi para los modelos 1 y 2 es del orden de $O(I \cdot J)$.

3.2.3 Entrenamiento de los modelos

Para entrenar los parámetros θ del modelo llevamos a cabo un estimación por máxima verosimilitud utilizando un corpus paralelo consistente en S pares de frases $\{(\mathbf{f}_s, \mathbf{e}_s) : s = 1, \ldots, S\}$, de modo que el valor óptimo de los parámetros vendrá dado una vez más por:

$$\hat{\theta} = \arg\max_{\theta} \prod_{s=1}^{S} \sum_{\mathbf{a}} p_\theta(\mathbf{f}_s, \mathbf{a} \mid \mathbf{e}_s) \ . \tag{3.17}$$

Esta estimación se realiza mediante el algoritmo EM [Baum 72, Dempster et al. 77]. Por ejemplo, para el modelo 1, en el paso de estimación, se calcula el valor esperado de que una palabra e sea traducción de otra f (a estos valores esperados se les llama contadores fraccionales). Dicho valor esperado para un par de frases $(\mathbf{f}, \mathbf{e})$ se calcula del siguiente modo:

$$c(f \mid e; \mathbf{e}, \mathbf{f}) = N(\mathbf{e}, \mathbf{f}) \cdot \sum_{\mathbf{a}} Pr(\mathbf{a} \mid \mathbf{e}, \mathbf{f}) \sum_j \delta(f, f_j) \delta(e, e_{a_j}) \ , \tag{3.18}$$

donde, $N(\mathbf{e}, \mathbf{f})$ es el número de veces que el correspondiente par aparece en el corpus, que normalmente valdrá 0 ó 1.

[2] En los sucesivo escribiremos $V(\mathbf{f}|\mathbf{e}; m)$ para denotar el alineamiento por Viterbi para el modelo m

En el paso de maximización, se obtienen los parámetros léxicos $t(f \mid e)$ que maximizan la verosimilitud del corpus de entrenamiento. Esto resulta en la siguiente reestimación:

$$t(f \mid e) = \frac{\sum_s c(f \mid e; \mathbf{f}^{(s)}, \mathbf{e}^{(s)})}{\sum_{s,f} c(f \mid e; \mathbf{f}^{(s)}, \mathbf{e}^{(s)})} \quad . \tag{3.19}$$

Dicho proceso se itera hasta un número máximo de iteraciones o hasta que la verosimilitud no aumenta en un cierto porcentaje.

Para el caso del modelo 2 habrá que hacer lo propio para los parámetros de alineamiento $a(\cdot)$, es decir:

$$c(i \mid j, J, I; \mathbf{e}, \mathbf{f}) = N(\mathbf{e}, \mathbf{f}) \cdot \sum_{\mathbf{a}} Pr(\mathbf{a} \mid \mathbf{e}, \mathbf{f}) \delta(i, a_j) \tag{3.20}$$

Y el valor de dichos parámetros que maximizan la verosimilitud serán:

$$a(i \mid j, J, I) = \frac{\sum_s c(i \mid j, J, I; \mathbf{f}^{(s)}, \mathbf{e}^{(s)})}{\sum_{s,j,J,I} c(i \mid j, J, I; \mathbf{f}^{(s)}, \mathbf{e}^{(s)})} \quad . \tag{3.21}$$

3.3 Modelos basados en fertilidades

Los modelos 1 y 2 se han definido realizando ciertas asunciones a las probabilidades condicionadas que aparecen en la ecuación 3.2. Esta ecuación es exacta, pero es sólo una de las muchas formas posibles en que la probabilidad conjunta de $\mathbf{f}$ y $\mathbf{a}$ se puede descomponer en una serie de productos de probabilidades condicionales. Cada término de este producto no es más que una forma de expresar matemáticamente el proceso generativo comentado en la sección 3.1 para obtener $\mathbf{f}$ y $\mathbf{a}$ a partir de $\mathbf{e}$.

Es de todos sabido que existen ciertas palabras que unas veces se traducen por una sola, por varias, o incluso por ninguna. Pues bien, al número de palabras que, de acuerdo a un determinado alineamiento entre un par de frases $(\mathbf{f}, \mathbf{e})$, se alinean con cada palabra e (o desde un punto de vista generativo, el número de palabras que e genera) se le conoce como fertilidad.

A diferencia de los modelos 1 y 2, los modelos 3, 4 y 5 modelan directamente la fertilidad de las palabras de una frase $\mathbf{e}$. Veamos como quedaría un modelo generativo para estos modelos: Dada una frase $\mathbf{e}$, en primer lugar habrá que elegir la fertilidad de cada una de las palabras que la forman y la lista de las palabras de $\mathbf{f}$ que cada una de ellas genera. A esta lista de palabras, que puede ser vacía, le llamamos *tupla*. Cada posible conjunto de tuplas, que llamaremos *tuplar*, será una variable aleatoria T[3]. Después de elegir el conjunto de tuplas

[3] Nótese que la tupla asociada a la i-ésima palabra de $\mathbf{e}$ será una variable aleatoria (T_i), y que la k-ésima palabra de la i-ésima tupla será una variable aleatoria (T_{ik}).

o *tuplar* (llamémosle $T = \boldsymbol{\tau}$), deberán permutarse convenientemente para finalmente obtener $\mathbf{f}$. Cada posible permutación también será una variable aleatoria, que le llamamos Π[4]. A la permutación que realmente nos dé lugar a $\mathbf{f}$ (es decir una instancia concreta de la variable Π) le llamaremos $\Pi = \boldsymbol{\pi}$.

Para un tuplar $\boldsymbol{\tau}$ y una permutación $\boldsymbol{\pi}$ en concreto, el modelo generativo lo podemos expresar matemáticamente cómo:

$$Pr(\boldsymbol{\tau}, \boldsymbol{\pi} \mid \mathbf{e}) = Pr(\phi \mid \mathbf{e}) \cdot Pr(\boldsymbol{\tau} \mid \phi, \mathbf{e}) \cdot Pr(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e}) \tag{3.22}$$

donde:

- $Pr(\phi \mid \mathbf{e})$ modela la probabilidad de que las palabras de $\mathbf{e}$ tengan una fertilidad en concreto y se define como:

$$Pr(\phi \mid \mathbf{e}) = \prod_{i=1}^{I} Pr(\phi_i \mid \phi_1^{i-1}, \mathbf{e}) \cdot Pr(\phi_0 \mid \phi_1^I, \mathbf{e}) \tag{3.23}$$

- $Pr(\boldsymbol{\tau} \mid \phi, \mathbf{e})$ modela la probabilidad de que las palabras de $\mathbf{e}$, conociendo sus fertilidades ϕ, generen el conjunto de tuplas $\boldsymbol{\tau}$, y se define como:

$$Pr(\boldsymbol{\tau} \mid \phi, \mathbf{e}) = \prod_{i=0}^{I} \prod_{k=1}^{\phi_i} Pr(\tau_{ik} \mid \tau_{i1}^{k-1}, \tau_0^{i-1}, \phi_0^I, \mathbf{e}) \tag{3.24}$$

- Y $Pr(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e})$ modela la probabilidad de que las palabras que forman el conjunto de tuplas $\boldsymbol{\tau}$ se ordenen de una forma específica $\boldsymbol{\pi}$, y se define como:

$$\begin{aligned} Pr(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e}) \quad = \quad & \prod_{i=1}^{I} \prod_{k=1}^{\phi_i} Pr(\pi_{ik} \mid \pi_{i1}^{k-1}, \pi_1^{i-1}, \tau_0^I, \phi_0^I, \mathbf{e}) \times \\ & \prod_{k=1}^{\phi_0} Pr(\pi_{0k} \mid \pi_{01}^{k-1}, \pi_1^I, \tau_0^I, \phi_0^I, \mathbf{e}) \end{aligned} \tag{3.25}$$

Y en general τ_{i1}^{k-1} representa la serie de valores $\tau_{i1}, \ldots, \tau_{ik-1}$; π_{i1}^{k-1} representa la serie de valores $\pi_{i1}, \ldots, \pi_{ik-1}$; y ϕ_i equivale a ϕ_{e_i}.

Como hemos comentado, conociendo $\boldsymbol{\tau}$ y $\boldsymbol{\pi}$ también conocemos $\mathbf{f}$ y $\mathbf{a}$, pero en general diferentes pares $(\boldsymbol{\tau}, \boldsymbol{\pi})$ pueden dar lugar al mismo par $(\mathbf{f}, \mathbf{a})$. El conjunto de pares $(\boldsymbol{\tau}, \boldsymbol{\pi})$ consistentes con $(\mathbf{f}, \mathbf{a})$, lo denotamos con $\langle \mathbf{f}, \mathbf{a} \rangle$ y se define como:

$$\langle \mathbf{f}, \mathbf{a} \rangle = \{(\boldsymbol{\tau}, \boldsymbol{\pi}) : f_{\pi_{ik}} = \tau_{ik}, a_{\pi_{ik}} = i; 0 \leq i \leq I, 1 \leq k \leq \phi_i\}$$

[4] Nótese que la posición en $\mathbf{f}$ de la k-ésima palabra de la i-ésima tupla será también una variable aleatoria, (Π_{ik})

De modo que el número de elementos de $\langle \mathbf{f}, \mathbf{a} \rangle$ viene dado por $\prod_{i=0}^{I} \phi_i$, dado que para cada τ_i hay $\phi_i!$ posibles formas de obtener el par $(\mathbf{f}, \mathbf{a})$.

En definitiva para los modelos basados en fertilidades tenemos que:

$$Pr(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) = \sum_{(\boldsymbol{\tau}, \boldsymbol{\pi}) \in \langle \mathbf{f}, \mathbf{a} \rangle} Pr(\boldsymbol{\tau}, \boldsymbol{\pi} \mid \mathbf{e}) \tag{3.26}$$

Antes de pasar a ver cada uno de los modelos basados en fertilidades propuestos en [Brown et al. 93] cabe comentar que en las distribuciones de probabilidad que aparecen en las ecuaciones (3.23), (3.24) y (3.25) existen gran cantidad de dependencias y por lo tanto un conjunto enorme de parámetros a estimar. Para hacer factible la estimación de estos modelos será necesario imponer una serie de restricciones a estos modelos para reducir considerablemente esta cantidad de parámetros, de modo que la estimación de los valores de esos parámetros se puede realizar de forma realista. Precisamente estas restricciones o asunciones a realizar en los modelos son las que van a diferenciar unos modelos de otros.

3.3.1 Los modelos 3, 4 y 5

La fórmula general para estos modelos es:

$$p_\theta(\boldsymbol{\tau}, \boldsymbol{\pi} \mid \mathbf{e}) = p_\theta(\phi \mid \mathbf{e}) \cdot p_\theta(\boldsymbol{\tau} \mid \phi, \mathbf{e}) \cdot p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e}) \tag{3.27}$$

$$p_\theta(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) = \sum_{(\boldsymbol{\tau}, \boldsymbol{\pi}) \in \langle \mathbf{f}, \mathbf{a} \rangle} p_\theta(\boldsymbol{\tau}, \boldsymbol{\pi} \mid \mathbf{e}) \tag{3.28}$$

En general, para todos estos modelos hacemos las siguientes asunciones:

- Para $p_\theta(\phi \mid \mathbf{e})$:
  - Que $Pr(\phi_i \mid \phi_1^{i-1}, \mathbf{e}) = n(\phi_i \mid e_i)$, es decir la probabilidad de que la palabra e_i tenga una fertilidad de ϕ_i solo va depender de la propia identidad de la palabra e_i.
  - Que $Pr(\phi_0 \mid \phi_1^I, \mathbf{e}) = n_0(\phi_0 \mid \sum_{i=1}^{I} \phi_i)$, es decir que la probabilidad de que la palabra nula e_0 tenga una fertilidad de ϕ_0 va a depender de la suma de las fertilidades de las palabras de e_1^I. Esto es tanto como decir que $\mathbf{f}$ tendrá ϕ_0 palabras espurias y $J - \phi_0$ no espurias (siendo J la longitud final de $\mathbf{f}$). Si consideramos p_1 como la probabilidad de generar una palabra espuria tras generar cada una de las $J - \phi_0$ palabras generadas por e_1^I, entonces $n_0(\phi_0 \mid J - \phi_0)$ se define como una distribución binomial de parámetro p_1 del siguiente modo:

    $$n_0(\phi_0 \mid J - \phi_0) = \binom{J - \phi_0}{\phi_0} p_0^{J - 2\phi_0} p_1^{\phi_0},$$

    donde $p_0 = 1 - p_1$ será la probabilidad de decidir no generar una palabra espuria. En definitiva de las $\binom{J-\phi_0}{\phi_0}$ posibilidades que tenemos de

generar esas palabras espurias, $p_1^{\phi_0}$ veces habremos decidido generar una palabra espuria y $p_0^{J-2\phi_0}$ veces no.

- Para $p_\theta(\boldsymbol{\tau} \mid \phi, \mathbf{e})$:
  - Que $Pr(\tau_{ik} \mid \tau_{i1}^{k-1}, \tau_0^{i-1}, \phi_0^I, \mathbf{e}) = t(\tau_{ik} \mid e_i)$, es decir que la probabilidad de que la palabra τ_{ik} de $\mathbf{f}$ sea generada por (o sea la traducción de) la palabra e_i solo dependerá de la propia e_i.
- Para $p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e})$:
  - Que $Pr(\pi_{0k} \mid \pi_{01}^{k-1}, \pi_1^I, \tau_0^I, \phi_0^I, \mathbf{e}) = \frac{1}{k}$, es decir la forma de colocar las palabras de $\mathbf{f}$ generadas por e_0 no depende de nada, por lo tanto el segundo término de la ecuación (3.25) será igual a $\frac{1}{\phi_0!}$, puesto que $\phi_0!$ denota las distintas posibles formas que tenemos de ordenar las ϕ_0 palabras generadas por e_0.

En definitiva tenemos que:

$$p_\theta(\phi \mid \mathbf{e}) = \prod_{i=1}^{I} n(\phi_i \mid e_i) \cdot n_0(\phi_0 \mid \sum_{i=1}^{I} \phi_i) \tag{3.29}$$

$$p_\theta(\boldsymbol{\tau} \mid \phi, \mathbf{e}) = \prod_{i=0}^{I} \prod_{k=1}^{\phi_i} t(\tau_{ik} \mid e_i) \tag{3.30}$$

La diferencia entre los modelos basados en fertilidades radica en las asunciones adicionales que se hacen para $p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e})$ con respecto al primer término de la ecuación (3.25). Veamos por separado cada uno de ellos.

El modelo 3

Adicionalmente para este modelo hacemos la siguiente asunción:

- Para $p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e})$:
  - Que $Pr(\pi_{ik} \mid \pi_{i1}^{k-1}, \pi_1^{i-1}, \tau_0^I, \phi_0^I, \mathbf{e}) = d(\pi_{ik} \mid i, J, I)$, es decir que la probabilidad de que la palabra τ_{ik} de $\mathbf{f}$ ocupe la posición π_{ik} dependerá solamente de la posición i en $\mathbf{e}$ de la palabra que la generó y de las longitudes de ambas frases.

De este modo, para el modelo 3 además tenemos que:

$$p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e}) = \frac{1}{\phi_0!} \prod_{i=1}^{I} \prod_{k=1}^{\phi_i} d(\pi_{ik} \mid i, J, I) \tag{3.31}$$

Por tanto los parámetros para este modelo serán

$$\theta = \{t(\cdot), n(\cdot), p_0, d(\cdot)\}$$

con:

$t(f \mid e)$	probabilidades de traducción
$n(\phi \mid e)$	probabilidades de fertilidad
p_0, p_1	probabilidades de fertilidad para e_0
$d(j \mid i, J, I)$	probabilidades de distorsión

donde $\phi = 0, 1, 2, \cdots \phi_{max}$

En la ecuación (3.31) el factor $1/\phi_0!$ da cuenta de la elección de la posiciones de las palabras generadas por e_0, es decir de π_{0k} con $k = 1, 2, \cdots, \phi_0$.

Las ecuaciones (3.27)-(3.31) describen el siguiente proceso generativo para obtener $\mathbf{f}$ o $fallo$ a partir de $\mathbf{e}$:

1. Para cada $i = 1, 2, \cdots, I$, elegir una longitud ϕ_i para τ_i (o lo que es lo mismo una fertilidad de ϕ_i para e_i) de acuerdo a la distribución de probabilidad $n(\phi_i \mid e_i)$.

2. Elegir una longitud ϕ_0 para τ_0 de acuerdo a la distribución $n_0(\phi_0 \mid \sum_{i=1}^{I} \phi_i)$.

3. Sea $J = \phi_0 + \sum_{i=1}^{I} \phi_i$.

4. Para cada $i = 0, 1, 2, \cdots, I$ y $k = 1, 2, \cdots, \phi_i$, elegir una palabra τ_{ik} de acuerdo a la distribución de probabilidad $t(\tau_{ik} \mid e_i)$.

5. Para cada $i = 0, 1, 2, \cdots, I$ y $k = 1, 2, \cdots, \phi_i$, elegir una posición π_{ik} de entre $1, \cdots, J$ de acuerdo a la distribución de probabilidad $d(\pi_{ik} \mid i, J, I)$.

6. Si alguna posición ha sido elegida más de una vez entonces retornar $fallo$.

7. Para cada $k = 1, 2, \cdots, \phi_0$, elegir una posición π_{0k} de entre las $\phi_0 - k + 1$ posiciones vacantes de acuerdo a la distribución de probabilidad uniforme (p_0).

8. Sea $\mathbf{f}$ la cadena con $f_{\pi_{ik}} = \tau_{\pi_{ik}}$.

A partir de las ecuaciones (3.29)-(3.31) se sigue que si $(\boldsymbol{\tau}, \boldsymbol{\pi})$ es consistente con $(\mathbf{f}, \mathbf{a})$ entonces:

$$p_\theta(\boldsymbol{\tau} \mid \phi, \mathbf{e}) = \prod_{j=1}^{J} t(f_j \mid e_{a_j}), \tag{3.32}$$

$$p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e}) = \frac{1}{\phi_0!} \prod_{j:a_j \neq 0} d(j \mid a_j, J, I) \ . \tag{3.33}$$

En esta última ecuación el productorio itera para todo $j = 1, 2, \cdots, J$ excepto cuando a_j sea igual a 0.

Sumando para todo par $(\boldsymbol{\tau}, \boldsymbol{\pi})$ consistente con $(\mathbf{f}, \mathbf{a})$ encontramos que

$$p_\theta(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) = \sum_{(\boldsymbol{\tau}, \boldsymbol{\pi}) \in \langle \mathbf{f}, \mathbf{a} \rangle} p_\theta(\boldsymbol{\tau}, \boldsymbol{\pi} \mid \mathbf{e}) \tag{3.34}$$

$$= n_0(\phi_0 \mid \sum_{i=1}^{I} \phi_i) \prod_{i=1}^{I} n(\phi_i \mid e_i) \cdot \phi_i! \times \prod_{j=1}^{J} t(f_j \mid e_{a_j}) \prod_{j:a_j \neq 0} d(j \mid a_j, J, I) \tag{3.35}$$

Los factores $\phi_i!$ que aparecen en la ecuación (3.35) son debidos a que existen $\prod_{i=0}^{l} \phi_i!$ términos igualmente probables en la suma de la fórmula (3.34). Del mismo modo el factor $\frac{1}{\phi_0!}$ queda anulado.

Este modelo es deficiente dado que

$$p_\theta(fallo \mid \boldsymbol{\tau}, \phi, \mathbf{e}) \equiv 1 - \sum_{\pi} p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e}) > 0 \ . \tag{3.36}$$

Es decir, cuando elegimos la posición $\pi_{ik} = j$ en la que pondremos la palabra f que está siendo generada por una palabra e_i no imponemos ninguna restricción, con lo que sería perfectamente válido asignar la misma posición a distintas palabras f. En otras palabras, el modelo de distorsión gasta cierta masa de probabilidad en asignar posiciones no válidas, por lo tanto (y así lo expresamos en la historia generativa de este modelo) será posible retornar $fallo$ en el proceso generativo.

Al igual que hemos comentado para el modelo 2, en este caso en el modelo de distorsión $d(j \mid i, J, I)$ se elimina la dependencia con el valor de I. Es decir utilizaremos como modelo de distorsión $d(j \mid i, J)$. En este caso esta simplificación será de especial interés a la hora de desarrollar algoritmos de búsqueda para este modelo. Esto es debido a que en el proceso de búsqueda la longitud de la cadena de salida es desconocida, por tanto la eliminación de la dependencia con I permitirá simplificar sustancialmente esos algoritmos de búsqueda.

En el modelo 3, al igual que en los anteriores, la elección de las posiciones de alineamiento entre palabras se realiza de forma absoluta, es decir la probabilidad de que una palabra f ocupe una posición determinada j no depende de las posiciones que ocupan las palabras previamente generadas. En definitiva todas las dependencias en el modelo de distorsión para el modelo 3 son de orden cero. En la figura 3.1 podemos ver un ejemplo de alineamiento para los modelos 1, 2 y 3, en el que no existen dependencias entre las posiciones alineadas, de ahí que este ejemplo valga para los 3 modelos mencionados.

El modelo 4

A diferencia del modelo 3 en este caso hacemos la asunción:

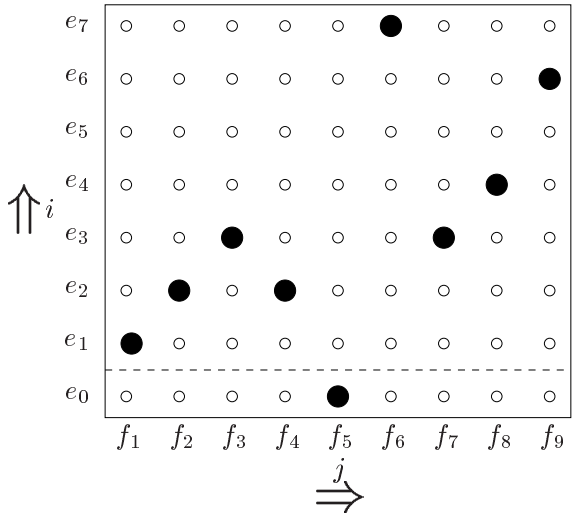


Figura 3.1: Dependencias en el alineamiento para los modelos 1, 2 y 3.

- Para $p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e})$:
  - Que $Pr(\pi_{ik} \mid \pi_{i1}^{k-1}, \pi_1^{i-1}, \tau_0^I, \phi_0^I, \mathbf{e}) = p_{ik}(\pi_{ik})$, donde:

$$p_{ik}(j) = \begin{cases} d_{=1}(j - c_{\rho_i} \mid \mathcal{E}_c(e_{\rho_i}), \mathcal{F}_c(\tau_{i1}) & \text{si } k = 1 \\ d_{>1}(j - \pi_{ik-1} \mid \mathcal{F}_c(\tau_{ik}) & \text{si } k > 1 \end{cases} . \tag{3.37}$$

En este caso se hace una asunción más realista incorporando una dependencia de primer orden, que a diferencia de los modelos anteriores establece dependencias relativas frente a absolutas con respecto a las posiciones de la frase. Además se distingue entre la posición a ocupar por la primera ($d_{=1}$) y el resto ($d_{>1}$) de palabras perteneciente a una tupla. Concretamente la posición a ocupar por τ_{i1} (primera palabra de la tupla τ_i), es decir π_{i1}, dependerá de: 1) las posiciones que ocuparon las palabras de la tupla (τ_{i-1}) generadas por la palabra anterior (e_{i-1}), y 2) de las clases (o categorías) a las que pertenecen las palabras origen y destino involucradas. Más específicamente se define ρ_i como la primera posición a la izquierda de i para la cual $\phi_i > 0$, y, en general, c_ρ es la función cielo de la media de las posiciones π_ρ ocupadas por las palabras de τ_ρ, es decir:

$$\rho_i = \max_{i' < i}\{i' : \phi_i > 0\}, \qquad c_\rho = \lceil \phi_\rho^{-1} \sum_{k=1}^{\phi_\rho} \pi_{\rho k} \rceil \ . \tag{3.38}$$

Las posiciones π_{ik}, con $k = 2 \cdots \phi_i$, a ocupar por cada τ_{ik} (resto de palabras de la tupla τ_i), dependerá de la posición π_{ik-1} ocupada por

la palabra previa (τ_{ik-1}) de esa tupla y por la clase de la palabra τ_{ik} de $\mathbf{f}$ en concreto que esté siendo generada.

En definitiva, para el modelo 4 además tenemos que:

$$p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e}) = \frac{1}{\phi_0!} \prod_{i=1}^{I} \prod_{k=1}^{\phi_i} p_{ik}(\pi_{ik}) \tag{3.39}$$

Al igual que el modelo 3, este modelo es deficiente dado que

$$p_\theta(fallo \mid \boldsymbol{\tau}, \phi, \mathbf{e}) \equiv 1 - \sum_{\pi} p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e}) > 0 \ . \tag{3.40}$$

Por tanto los parámetros para este modelo son

$$\theta = \{t(\cdot), n(\cdot), p_0, d_{=1}(\cdot), d_{>1}(\cdot)\}$$

con:

$t(f \mid e)$	probabilidades de traducción
$n(\phi \mid e)$	probabilidades de fertilidad
p_0, p_1	probabilidades de fertilidad para e_0
$d_{=1}(\Delta j \mid \mathcal{E}_c, \mathcal{F}_c)$	probabilidades de distorsión para la primera palabra de una tupla
$d_{>1}(\Delta j \mid \mathcal{F}_c)$	probabilidades de distorsión para el resto de palabras de una tupla

donde Δj es un entero y $\mathcal{E}_c(e)$ y $\mathcal{F}_c(f)$ son las clases/categorías a las que pertenecen las palabras e y f respectivamente.

Las ecuaciones (3.27)-(3.30) y (3.39) describen el siguiente proceso generativo para obtener $\mathbf{f}$ o $fallo$ a partir de $\mathbf{e}$:

1. Elegir un conjunto de tuplas $\boldsymbol{\tau}$ a partir de los pasos 1–4 para el modelo 3.
2. Para cada $i = 1, 2, \cdots, I$ y $k = 1, 2, \cdots, \phi_i$, elegir una posición π_{ik} como sigue:
   - Si $k = 1$ entonces elegir π_{i1} de acuerdo a la distribución: $d_{=1}(\pi_{i1} - c_{\rho_i} \mid \mathcal{E}_c(e_{\rho_i}), \mathcal{F}_c(\tau_{i1}))$.
   - Si $k > 1$ entonces elegir π_{ik} de acuerdo a la distribución: $d_{>1}(\pi_{ik} - \pi_{ik-1} \mid \mathcal{F}_c(\tau_{ik}))$.
3. Acabar generando $\mathbf{f}$ siguiendo los pasos 6–8 del modelo 3.

A partir de las ecuaciones (3.29), (3.30) y (3.39) y sumando para todo par $(\boldsymbol{\tau}, \boldsymbol{\pi})$ consistente con $(\mathbf{f}, \mathbf{a})$ encontramos que:

$$\begin{aligned} p_\theta(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) &= \sum_{(\boldsymbol{\tau},\boldsymbol{\pi}) \in \langle \mathbf{f}, \mathbf{a} \rangle} p_\theta(\boldsymbol{\tau}, \boldsymbol{\pi} \mid \mathbf{e}) \\ &= n_0(\phi_0 \mid \sum_{i=1}^{I} \phi_i) \cdot \prod_{i=1}^{I} n(\phi_i \mid e_i) \cdot \prod_{j=1}^{J} t(f_j \mid e_{a_j}) \times \\ &\quad \prod_{j:a_j \neq 0} \begin{cases} d_{=1}(j - c_{\rho_{a_j}} \mid \mathcal{E}_c(e_{\rho_{a_j}}), \mathcal{F}_c(f_j)) & \text{si } j = h(a_j) \\ d_{>1}(j - p(j) \mid \mathcal{F}_c(f_j)) & \text{si } j \neq h(a_j) \end{cases} \end{aligned} \tag{3.41}$$

donde:

- $h(i)$ es la posición con la que se alinea la primera palabra de la tupla i:

$$h(i) = \min_k \{k : i = a_k\}$$

- y $p(j)$ la posición que ocupa la palabra previa a la que ocupa la posición a_j en la misma tupla:

$$p(j) = \max_{k<j} \{k : a_j = a_k\}$$

Nótese que a diferencia del modelo 3, en este caso no aparece el factor de $\phi_i!$ dado el orden que impone el modelo en la forma en que se generan las palabras pertenecientes a una tupla en concreto.

Para clarificar si cabe un poco más la notación, para este modelo (al igual que ocurrirá con el modelo 5) las dependencias en el modelo de distorsión son de primer orden, es decir la elección de las posiciones a ocupar por las palabras generadas se hará de forma relativa a las posiciones previamente generadas. En otras palabras, la elección de la posición j a ocupar por una palabra f, que esté siendo generada por la palabra e_i, dependerá de las posiciones que ocupan las palabras generadas por la palabra destino previa, es decir e_{i-1}. Un ejemplo de las dependencias en el alineamiento para este modelo las podemos ver en la figura 3.2. Por ejemplo, la elección de la posición de la palabra f_3 (generada por e_3) depende de f_2 y f_4, palabras que han sido generadas por e_2. En este modelo también se establece la diferencia entre la primera palabra que forma una tupla ($d_{=1}$) y el resto de las que conforman esa tupla ($d_{>1}$). Siguiendo con el ejemplo de la figura 3.2, la posición de la palabra f_7 depende solamente de la posición que ocupa f_3, es decir la segunda palabra de la tupla generada por e_3 sólo depende la palabra previamente generada dentro de esa tupla. Con todo esto podemos concluir por tanto que el valor de Δ_j para $d_{=1}$, podrá tomar valores negativos (como es el caso de f_6 en el ejemplo), y para $d_{>1}$ sólo podrá tomar valores estrictamente positivos. Por otra parte el resto de dependencias

en el modelo de distorsión se definen con respecto a clases de palabras en vez de con respecto a la identidad de ellas. Esto ayudará a aliviar el problema típico de falta de datos de entrenamiento y hará que la estimación de los parámetros sea más robusta.

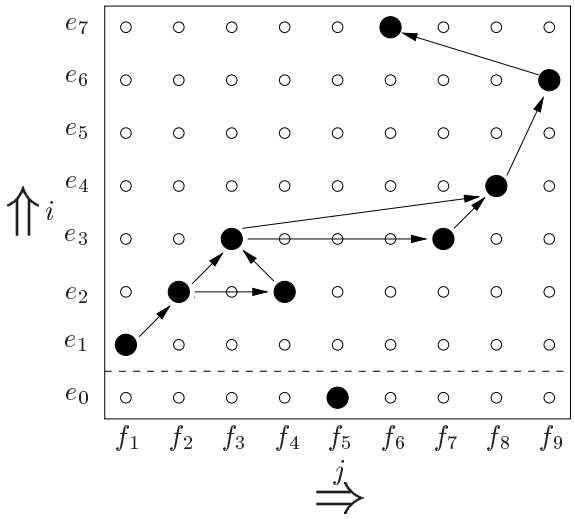


Figura 3.2: Ejemplo de dependencias en el alineamiento para el modelo 4.

El modelo 5

En este modelo, a diferencia de los anteriores, hacemos la asunción:

$$p_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \phi, \mathbf{e}) = \frac{1}{\phi_0!} \prod_{i=1}^{I} \prod_{k=1}^{\phi_i} p_{ik}(\pi_{ik}) \tag{3.42}$$

donde:

$$p_{ik}(j) = \epsilon_{ik}(j) \begin{cases} d_{=1}(\Delta_j \mid \mathcal{F}_c(\tau_{i1}), \nu_{i1}(J) - \phi_i + k) & \\ & \text{si } k = 1 \\ d_{>1}(\Delta_j \mid \mathcal{F}_c(\tau_{ik}), \nu_{ik}(J) - \nu_{ik}(\pi_{ik-1}) - \phi_i + k) & \\ & \text{si } k > 1 \end{cases} . \tag{3.43}$$

y:

$$\Delta_j = \begin{cases} \nu_{i1}(j) - \nu_{i1}(c_{\rho_i}) & \text{si } k = 1 \\ \nu_{ik}(j) - \nu_{ik}(\pi_{ik-1}) & \text{si } k = 1 \end{cases}$$

En la ecuación (3.43), ρ_i es la primera posición a la izquierda de i cuya fertilidad es distinta de cero, y c_ρ es la función cielo de la media de las posiciones

de las palabras de τ_ρ (ver ecuación (3.38)); $\epsilon_{ik}(j)$ es 1 si la posición j está vacante una vez que todas las palabras de las tuplas $i' < i$ y las primeras $k-1$ palabras de la tupla i han sido ya alineadas, y 0 en otro caso; $\nu_{ik}(j)$ es el número de posiciones vacantes hasta la posición j, es decir:

$$\nu_{ik}(j) = \sum_{j' \leq j} \epsilon_{ik}(j')$$

Como podemos ver en la figura 3.3 las dependencias de este modelo son prácticamente idénticas a las del modelo 4, sólo que en este caso Δ_j establece la diferencia entre las posiciones vacantes hasta la posición a ocupar por la palabra actual y las vacantes hasta las palabras previamente generadas. En la parte derecha de la figura vemos cómo se van cubriendo las posiciones durante el proceso generativo. Con respecto al resto de dependencias, para el caso de $d_{=1}$ se ha eliminado la dependencia con la clase de la palabra e previa, y se ha introducido una dependencia, al igual que para el caso de $d_{>1}$, con las vacantes que quedan libres para colocar las palabras de la tupla actual, es decir la tupla que está siendo generada.

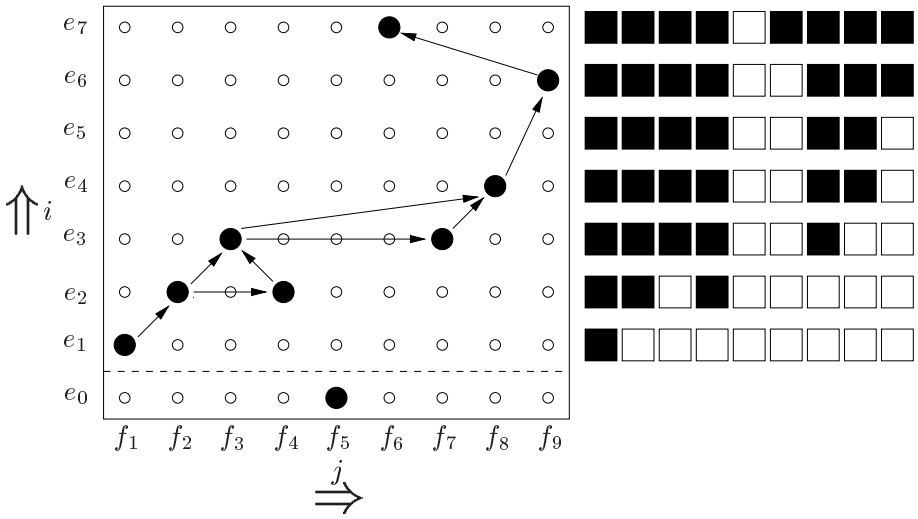


Figura 3.3: Ejemplo de dependencias en el alineamiento para el modelo 5.

Por tanto los parámetros para este modelo son

$$\theta = \{t(\cdot), n(\cdot), p_0, d_{=1}(\cdot), d_{>1}(\cdot)\}$$

con:

$t(f \mid e)$	probabilidades de traducción
$n(\phi \mid e)$	probabilidades de fertilidad
p_0, p_1	probabilidades de fertilidad para e_0
$d_{=1}(\Delta j \mid \mathcal{F}_c, \nu)$	probabilidades de distorsión para la primera palabra de una tupla
$d_{>1}(\Delta j \mid \mathcal{F}_c, \nu)$	probabilidades de distorsión para el resto de palabras de una tupla

Aquí $\nu = 1, 2, \cdots J$.

Este modelo no es deficiente dado que precisamente la variable ν lleva en cuenta las posiciones de la cadena de entrada que ya han sido generadas. Es precisamente esta la gran diferencia de este modelo con respecto al modelo 4.

Las ecuaciones (3.27)-(3.30) y (3.43) describen el siguiente proceso generativo para obtener $\mathbf{f}$ a partir de $\mathbf{e}$:

1. Elegir un conjunto de tuplas $\boldsymbol{\tau}$ a partir de los pasos 1–4 para el modelo 3.
2. Para cada $i = 1, 2, \cdots, I$ y $k = 1, 2, \cdots \phi_i$, elegir una posición π_{ik} como sigue:
   - Si $k = 1$ entonces elegir π_{i1} de acuerdo a la distribución: $d_{=1}(\nu_{i1}(j) - \nu_{i1}(c_{\rho_i}) \mid \mathcal{F}_c(\tau_{i1}), \nu_{i1}(J) - \phi_i + k)$
   - Si $k > 1$ entonces elegir π_{ik} de acuerdo a la distribución: $d_{>1}(\nu_{ik}(j) - \nu_{ik}(\pi_{ik-1}) \mid \mathcal{F}_c(\tau_{ik}), \nu_{ik}(J) - \nu_{ik}(\pi_{ik-1}) - \phi_i + k)$
3. Acabar generando $\mathbf{f}$ siguiendo los pasos 7–8 del modelo 3.

3.3.2 Alineamiento por Viterbi

En este caso, a diferencia de los modelos anteriores, la maximización sobre todos los posibles alineamientos no es descomponible, con lo que no conocemos ningún algoritmo eficiente que calcule el alineamiento por Viterbi, utilizando por ejemplo técnicas de programación dinámica, que realice el cálculo en tiempos de orden polinomial. La solución pasaría por utilizar algoritmos complicados basados en la técnica de ramificación y poda (como el $A^\star$), pero en realidad se utiliza un algoritmo voraz muy eficiente como se indica en [Brown et al. 93].

Este algoritmo voraz sigue la técnica de ascenso de colina (*hillclimbing*) y funciona del siguiente modo: se parte del alineamiento por Viterbi para un modelo más sencillo, por ejemplo el modelo 2. A este alineamiento se le aplican una serie de perturbaciones para construir un conjunto de vecinos. Se elige el mejor alineamiento de ese conjunto y se itera el proceso hasta que no se obtengan mejores alineamientos.

Para clarificar un poco más veamos en que consiste el concepto de *vecino* de un alineamiento. Se dice que dos alineamientos $\mathbf{a}$ y $\mathbf{a}'$ difieren en un *movimiento* si hay exactamente un valor de j para el cual $a_j \neq a'_j$. Se dice que difieren en un *intercambio* si para todo $j : a_j = a'_j$ excepto para dos valores, j_1 y j_2, para

los cuales $a_{j_1} = a'_{j_2}$ y $a_{j_2} = a'_{j_1}$. Se dice que que dos alineamientos son *vecinos* si son idénticos o difieren a lo sumo por un *movimiento* o un *intercambio*.

Sea $b(\mathbf{a})$ el mejor vecino de $\mathbf{a}$, es decir para el cual es máxima la verosimilitud $Pr(\mathbf{a} \mid \mathbf{f}, \mathbf{e})$. De este modo la secuencia $\mathbf{a}, b(\mathbf{a}), b^2(\mathbf{a}) \equiv b(b(\mathbf{a})), \cdots$, converge en un número finito de pasos a un alineamiento que llamaremos $b^\infty(\mathbf{a})$. De hecho si el alineamiento $\mathbf{a}$ del que partimos es el alineamiento por Viterbi para el modelo 2 ($V(\mathbf{f} \mid \mathbf{e}; 2)$) entonces $b^\infty(V(\mathbf{f} \mid \mathbf{e}; 2))$ será una aproximación buena a $V(\mathbf{f} \mid \mathbf{e}; 3)$.

Según la definición de vecindad, obtener $V(\mathbf{f} \mid \mathbf{e}; 3)$ es bastante sencillo y rápido debido a que el modelo de distorsión para este modelo es simple. Por esta misma razón es mucho más costoso de calcular el alineamiento por Viterbi para los modelos 4 y 5, aunque se construyen de forma análoga (para más detalles acerca del cálculo de $V(\mathbf{f} \mid \mathbf{e}; 4)$ y $V(\mathbf{f} \mid \mathbf{e}; 5)$ ver [Brown et al. 93]).

Después de todo esto sería erróneo llamar alineamiento por Viterbi al mejor alineamiento posible, entre dos frases, para los modelos basados en fertilidades. De todos modos, a partir de ahora, abusaremos del lenguaje y llamaremos alineamiento por Viterbi al mejor alineamiento que seamos capaces de obtener.

3.3.3 Entrenamiento de los modelos

Para el entrenamiento de los modelos basados en fertilidades (es decir, modelos 3, 4 y 5), se sigue un proceso similar al expuesto anteriormente para los modelos 1 y 2, pero teniendo en cuenta además los modelos de fertilidad y distorsión asociados. Para más detalles en este aspecto remitimos al lector a [Brown et al. 93].

3.4 Otros modelos de alineamiento

En la bibliografía sobre la materia existen otros modelos estadísticos de alineamiento, los cuales están fuertemente inspirados en los modelos de IBM o se basan en algunos de ellos. En esta sección comentaremos brevemente un modelo más, ampliamente utilizado en otros sistemas de traducción automática. Existen otros tipos de modelos de alineamiento basados en palabras como [Dagan et al. 93, Kay and Röscheisen 93, Chang and Chen 94, Fung and McKeown 94, Jones and Somers 95, Macklovitch and Hannan 96, Vogel et al. 96, Melamed 97, Ker and Chang 97, Och et al. 99, Huang and Choi 00, Och and Ney 00b, Tillmann 01, García-Varea et al. 02b].

Por otra parte existen modelos de alineamiento basados en grupos de palabras. De entre ellos podemos destacar los propuestos en [Och 02, Wang and Waibel 98, Tomás and Casacuberta 01, Vilar 98].

También se han realizado trabajos en los que se han evaluado la calidad de modelos de alineamiento basados en palabras o comparaciones entre distintos modelos de alineamiento. Dentro de estos trabajos cabe destacar [Shin et al. 96,

Wang and Waibel 98, Ahrenberg et al. 00, Och and Ney 00a, Och and Ney 03, Och 02].

3.4.1 El modelo HMM

Otro modelo ampliamente utilizado es un modelo basado en alineamientos monótonos, conocido por modelo HMM y propuesto en [Vogel et al. 96], el cual exponemos con algo más de detalle que los comentados previamente pues será utilizado dentro de alguno de los algoritmos de búsqueda que desarrollaremos más adelante.

Partiendo de la descomposición 3.2 se diferencia con el modelo 2 en que se asume una dependencia de primer orden para los alineamientos a_j, de modo que se hace la asunción:

$$Pr(a_j \mid f_1^{j-1}, a_1^{j-1}, \mathbf{e})) = h(a_j \mid aj-1, I)$$

de modo que se obtiene la siguiente descomposición para $p_\theta(f_1^J \mid e_1^I)$:

$$p_\theta(f_1^J \mid e_1^I) = \epsilon(J \mid I) \cdot \sum_{a_1^J} \prod_{j=1}^{J} h(a_j \mid a_{j-1}, I) \cdot t(f_j \mid e_{a_j}) \tag{3.44}$$

con probabilidad de alineamiento $h(i \mid i', I)$ y probabilidad de traducción $t(f \mid e)$ idéntica al resto de modelos.

Para hacer que las probabilidades de alineamiento no dependan de posiciones absolutas se asume que $h(i \mid i', I)$ solo depende de la diferencia $(i - i')$, por tanto usando un conjunto de parámetros $\{c(i - i')\}$, la probabilidad de alineamiento se puede reescribir como:

$$h(i \mid i', I) = \frac{c(i - i')}{\sum_{i''=1}^{I} c(i'' - i')} \tag{3.45}$$

Esta definición asegura que las probabilidades de alineamiento satisfacen la restricción de normalización para cada i', $i' = 1, \ldots, I$. A este modelo los autores le llaman modelo HMM homogéneo [Vogel et al. 96]. Una idea similar es la que se expone en [Dagan et al. 93].

En esta formulación, a diferencia del modelo 2, no se tiene en cuenta la palabra vacía e_0. En [Och 02] se propone una extensión para tenerla en cuenta.

Al igual que para los modelos 1 y 2 el alineamiento por Viterbi para este modelo se puede calcular de forma exacta. Para ello se utiliza un algoritmo basado en programación dinámica, descrito en [Vogel et al. 96], con un coste de orden de $O(I^2 \cdot J)$.

3.5 Esquema de entrenamiento

Cómo hemos visto en la sección anterior cada modelo de IBM (1–5) se puede ver como un refinamiento del anterior, de modo que conforme vamos aumentando de modelo las asunciones que se hacen son más realistas.

En principio podríamos pensar que para entrenar cualquiera de los modelos anteriores bastaría con inicializar uniformemente todos sus parámetros y realizar un número determinado de iteraciones del algoritmo EM hasta que se alcance un criterio de parada.

El hecho de que un modelo superior se base, total o parcialmente, en su inmediato predecesor (por ejemplo, los parámetros léxicos $t(\cdot)$ forman parte de todos los modelos), hace pensar en utilizar la información que proporciona el modelo anterior para establecer una inicialización razonable de los parámetros del modelo actual. El problema radica en como se puede establecer esta transferencia de parámetros entre modelos.

Lo más razonable parece ser empezar por los modelos más sencillos, por ejemplo los modelos 1 o 2, e ir aumentando de modelo progresivamente. Además, el hecho de que para los modelos 1 y 2 sea posible realizar la suma para todo posible alineamiento entre un par de frases (ver ecuación (3.1)) garantiza que, inicializando sus parámetros a cualquier valor distinto de cero, se alcance un máximo global. Este hecho, erige a estos modelos como fuertes candidatos a realizar la estimación de sus parámetros en primer lugar. De este modo, inicializar los parámetros $t(\cdot)$ del modelo 2 a los previamente estimados del modelo 1, no significa que se vaya a obtener una mejor estimación de sus parámetros, pero si que la convergencia del algoritmo EM sea mucho más rápida.

Para el resto de modelos, el problema se agrava pues no existe forma práctica de realizar la suma para todo posible alineamiento entre un par dado de frases. Por ello se apoyan en los modelos inferiores para realizar una estimación inicial de sus parámetros. Por ejemplo, para el modelo 3 los parámetros $t(\cdot)$ y $d(\cdot)$ se inicializan directamente a partir de los parámetros $t(\cdot)$ y $a(\cdot)$ estimados para el modelo 2; en cuanto a los parámetros $n(\cdot)$ existen hasta cuatro formas distintas de inicializarlos (ver [Knight 99b]). Del mismo modo podemos transferir los parámetros del modelo 3 al modelo 4 y los del 4 al 5.

El mismo problema antes comentado de la imposibilidad de llevar a cabo la suma para todo alineamiento en los modelos basados en fertilidad, hace que el conjunto de alineamientos más probables entre un par de frases se construya en base al alineamiento por Viterbi del modelo inmediatamente inferior, utilizando un método de ascenso de colina prácticamente idéntico al descrito para obtener el alineamiento por Viterbi para estos modelos. Esto conlleva que tras cada iteración del algoritmo EM para estos modelos, además de actualizar la estimación de sus parámetros, tenga que realizarse una actualización de los parámetros del modelo en el que se basa para construir el conjunto de alineamientos más probables. En otras palabras, que tras una iteración del modelo 3 habrá que

actualizar también los parámetros del modelo 2 pues es en ellos en los que se basará en la siguiente iteración.

En definitiva llamamos esquema de entrenamiento al conjunto de iteraciones del algoritmo EM que realizamos de cada modelo, al orden en que se realizan y a la transferencia de parámetros entre ellos. A partir de ahora y mientras no digamos lo contrario la transferencia de parámetros entre modelos se realizará de la forma que se describe en [Brown et al. 93].

3.6 Resumen

En este capítulo hemos expuesto, con cierto nivel de detalle, los modelos de traducción que serán la base de toda la experimentación que llevaremos a cabo en el resto de capítulos. Hemos intentado ser rigurosos a la vez que intuitivos en la descripción de los modelos de traducción, entre otras cosas para suavizar la barrera que conlleva la complejidad matemática de los modelos estadísticos de traducción, y que a veces hace poco atractiva esta aproximación a la traducción automática.

Concretamente, hemos expuesto la familia de modelos estadísticos de alineamiento de IBM y el modelo HMM, así como una pequeña recensión bibliográfica de otros modelos (basados en palabras o en grupos de palabras) ampliamente utilizados en otros trabajos.

También hemos comentado conceptos como el de historia o proceso generativo, alineamiento por Viterbi (o alineamiento más probable), y el de esquema de traducción.

Capítulo 4

Modelos de traducción basados en Máxima Entropía

El trabajo expuesto en este capítulo se puede considerar como una continuación al trabajo de Adam L. Berger y colaboradores [Berger et al. 96b], en donde se introducen los modelos léxicos por máxima entropía y cómo utilizarlos, de forma desacoplada, en sistema estadísticos de traducción automática. En primer lugar introduciremos la aproximación por máxima entropía mediante la que definiremos modelos léxicos de traducción refinados. Veremos cómo integrar estos modelos de forma eficiente dentro del algoritmo de entrenamiento por máxima verosimilitud para los modelos convencionales y haremos un estudio empírico de la calidad de los modelos y de los alineamientos obtenidos con esta aproximación comparándolos con los resultados obtenidos con modelos convencionales. Cabe destacar que en este capítulo empiezan las aportaciones de esta tesis.

4.1 Introducción

Intuitivamente el principio en que se basa la máxima entropía es simple: modelar todo lo conocido y no asumir nada acerca de lo desconocido. En otras palabras, dada una colección de hechos, elegir el modelo que es consistente con todos los hechos, pero por contra que sea lo más uniforme posible.

El concepto de máxima entropía tiene una historia muy larga. El hecho de elegir la hipótesis más simple posible está contemplado en la "Navaja de Occam" ("Pluralitas non est ponenda sine necessitate"), e incluso aparece anteriormente

en la Biblia [Jaynes 90]. Laplace se puede considerar como el padre de la máxima entropía cuando enunció: "Cuando no hay información para distinguir entre la probabilidad de dos eventos, la mejor estrategia es considerarlos igualmente probables" [Guiasu and Shenitzer 94], en su "Principle of Insufficient Reason". Lo mismo afirmó E.T.Jaynes, un pionero más reciente de la máxima entropía, [Jaynes 90]:

> "*... the fact that a certain probability distribution maximizes entropy subject to certain constraints representing out incomplete information, is the fundamental property which justifies the use of that distribution for inference; it agrees with everything that is known, but carefully avoids anything that is not known. It is a transcription into mathematics of an ancient principle of wisdom ...*".

La primera aplicación de la máxima entropía se realizó en física durante el último lustro del siglo XIX y las dos primeras décadas del siglo XX, ampliamente utilizado en termodinámica estadística en sus dos variantes: la clásica y mecánica cuántica (Maxwell-Boltzmann, Fermi-Dirac, Bose-Einstein). También aparece a finales de los años 40 cuando Shannon enunció su teoría de la información y la entropía [Cover and Thomas 91] y en la misma época los modelos log-lineales [Bishop et al. 75] (estrechamente relacionados con los modelos por máxima entropía) fueron ampliamente usados en análisis multivariado discreto y tablas de contingencia. Como ya hemos comentado, a finales de los años 50 fue cuando E.T.Jaynes propuso el principio de máxima entropía (también expuesto en [Guiasu and Shenitzer 94]) y junto con algunos otros estadísticos abogaron por la estimación general de distribuciones de probabilidad, lo cual iba más allá de la estimación estadística clásica, y no fue aceptada por algunos estadísticos ortodoxos. Conforme pasaron los años, el principio de máxima entropía, se aplica con éxito en el área del procesamiento de la señal como por ejemplo en estimación espectral (LPC, coeficientes lineales predictivos; y AR, modelos autoregresivos), restauración de imágenes, estimación general de distribuciones de probabilidad, etc.

Los primeros trabajos en los que se utilizan modelos basados en máxima entropía en el ámbito del procesamiento del lenguaje natural datan de principios/mediados de la década pasada, aunque realmente se han puesto de moda hace tan sólo unos años. La mayoría de los trabajos en esta línea se han dedicado al problema del modelado de lenguaje, por ejemplo a principios de los años 90 IBM utilizó esta filosofía por primera vez. A partir de ahí se han realizado gran cantidad de trabajos, como por ejemplo [Della Pietra et al. 94, Rosenfeld 96, Simons et al. 97] o en [Wu 98, Khudanpur and Wu 99, Martin et al. 99, Peters and Klakow 99, Wu and Khudanpur 99]. También cabe destacar, más recientemente, a [Khudanpur and Wu 00, Martin et al. 00, Wu and Khudanpur 00, Wu and Khudanpur 02].

Otros autores aplican la máxima entropía a distintos problemas en

procesamiento del lenguaje natural como análisis sintáctico [Charniak 99, Ratnaparkhi 99], etiquetado POS [Ratnaparkhi 96] y resolución de ambigüedades léxicas [Ratnaparkhi 98]. En [Ratnaparkhi 97] podemos encontrar una introducción general al procesamiento del lenguaje natural usando esta técnica.

Ya en el ámbito de la traducción automática podemos encontrar algunos trabajos de cierta relevancia. En [Berger et al. 96b] esta aproximación se aplica al sistema Candide de IBM para crear modelos dependientes del contexto para particionar frases de forma automática y mejorar el reordenamiento de palabras en traducción. Técnicas similares se utilizan en [Papineni et al. 98] y en [Foster 00b] para modelos de traducción directos al contrario de los que se exponen en [Brown et al. 93]. En [Foster 00a] se describen dos métodos para incorporar información acerca de la posición relativa de pares de palabras bilingües; y en [García-Varea et al. 01] se utilizan modelos por máxima entropía para reducir la perplejidad de los modelos y los errores de traducción en base a un algoritmo de repuntuación de hipótesis. Por último comentar que en [Och and Ney 02] se presenta una aproximación alternativa al modelo clásico de la fuente y el canal para atacar el problema de la traducción automática estadística, en la cual se pueden incorporar diversas fuentes de conocimiento.

4.2 Estimación de modelos de traducción por máxima entropía

Para introducir el concepto de máxima entropía en el ámbito de la traducción automática utilizaremos un ejemplo similar al expuesto en [Berger et al. 96b] dada su gran sencillez e intuitiva exposición.

Supongamos que queremos modelar las decisiones de un experto traductor acerca de la traducción de la palabra inglesa *in* al castellano. Nuestro modelo p_{in}, de las decisiones del experto, asigna a cada palabra (o frase) en castellano f una estimación $p_{in}(f)$ de la probabilidad de que el experto elegiría f como traducción de *in*. Supongamos que para obtener $p_{in}(f)$ recopilamos una muestra con gran cantidad de ejemplos de decisiones tomadas por el experto. Nuestro objetivo entonces es extraer una serie de hechos acerca del proceso de decisión a partir de la muestra (primera tarea en modelado) que nos ayude a construir el modelo de ese proceso de decisión (segunda tarea en modelado).

Una buena pista que podemos extraer de la muestra es la lista de posibles traducciones de la palabra en cuestión. Por ejemplo, podríamos observar que el experto siempre elige una de las siguientes palabras (frases) en castellano: {*dentro*, *en*, *por*, *a lo largo de*, *durante*}. Con esta información podemos imponer una primera restricción a nuestro modelo:

$$p_{in}(\textit{dentro}) + p_{in}(\textit{en}) + p_{in}(\textit{por}) + p_{in}(\textit{a lo largo de}) + p_{in}(\textit{durante}) = 1$$

Esta ecuación representa nuestro primer estadístico del proceso, ahora podemos proceder a buscar un modelo que se ajuste a esta ecuación. Evidentemente hay

infinitos modelos que cumplen esta ecuación. Uno podría ser aquel que asigne a $p_{in}(dentro) = 1$, es decir, este modelo siempre predice *dentro* como traducción de *in*. Otro modelo podría ser aquel que asigne una probabilidad de 0.5 a *dentro* y de 0.5 a *en*. Evidentemente cualquiera de estos dos modelos son malos pues asumen más de lo que realmente conocemos acerca del proceso de decisión del experto. Todo lo que nosotros sabemos es que el experto elige una de las cinco posibles traducciones y nada más, de modo que el modelo más intuitivo que se nos puede ocurrir a priori sería el que contenga la restricción adicional:

$$p_{in}(\textit{dentro}) = p_{in}(\textit{en}) = p_{in}(\textit{por}) = p_{in}(\textit{a lo largo de}) = p_{in}(\textit{durante}) = 1/5$$

Este modelo es el más uniforme de acuerdo a lo que conocemos, aunque no el más uniforme de todos los modelos posibles, el cual sería aquel que asigne la misma probabilidad a toda posible palabra (frase) en castellano.

De todas formas, de nuestra muestra podemos obtener más pistas acerca del proceso de decisión del experto. Supongamos que el experto elige *dentro* y *en* en el 30% de los casos. Podríamos añadir este conocimiento extra a nuestro modelo mediante una nueva restricción. Por tanto, el modelo tendrá que satisfacer las dos restricciones impuestas, es decir:

$$p_{in}(\textit{dentro}) + p_{in}(\textit{en}) = 3/10$$
$$p_{in}(\textit{dentro}) + p_{in}(\textit{en}) + p_{in}(\textit{por}) + p_{in}(\textit{a lo largo de}) + p_{in}(\textit{durante}) = 1$$

Al igual que antes hay muchas distribuciones de probabilidad que satisfacen estas restricciones. Sin tener en cuenta otro tipo de información, una elección razonable (del mismo tipo que la anterior) sería elegir aquel modelo $p_{in}(\cdot)$ que distribuya su masa de probabilidad uniformemente, o sea, aquella distribución que asigne su probabilidad de forma equitativa sujeta a las restricciones impuestas. Este modelo asignaría:

$$p_{in}(\textit{dentro}) = p_{in}(\textit{en}) = 3/20$$
$$p_{in}(\textit{por}) = p_{in}(\textit{a lo largo de}) = p_{in}(\textit{durante}) = 7/30$$

Supongamos que inspeccionamos una vez más la muestra y que observamos un hecho interesante: que en la mitad de los casos el experto elige *dentro* o *por*. Podemos incorporar esta información en nuestro modelo como una tercera restricción, esto es:

$$p_{in}(\textit{dentro}) + p_{in}(\textit{en}) = 3/10$$
$$p_{in}(\textit{dentro}) + p_{in}(\textit{en}) + p_{in}(\textit{por}) + p_{in}(\textit{a lo largo de}) + p_{in}(\textit{durante}) = 1$$
$$p_{in}(\textit{dentro}) + p_{in}(\textit{por}) = 1/2$$

Una vez más podemos buscar la distribución más uniforme que satisfaga las restricciones, sólo que ahora la elección no es tan obvia como en los casos anteriores. Conforme hemos añadido complejidad al modelo nos hemos encontrado

con dos problemas en uno. Primero, ¿que significa exactamente "uniforme" y como podemos medir la uniformidad de un modelo?, y segundo, conociendo la respuesta a la pregunta anterior, ¿cómo encontrar el modelo más uniforme que cumpla una serie de restricciones como las comentadas?

El método de máxima entropía tiene la respuesta a estas preguntas. Como ya hemos comentado, intuitivamente el principio de ME es simple: dada una colección de hechos, elegir el modelo que es consistente con todos los hechos, pero por contra que sea lo más uniforme posible. Esta es precisamente la aproximación que hemos seguido para nuestro modelo p_{in} en el ejemplo anterior.

4.2.1 Modelado por máxima entropía

Consideremos un proceso aleatorio que produce un valor de salida f, un elemento de un conjunto finito $\mathcal{F}$. Para el ejemplo de traducción que hemos considerado, el proceso genera una traducción de la palabra *in*, y la salida f puede ser cualquier palabra del conjunto {*dentro*, *en*, *por*, *a lo largo de*, *durante*}. Al generar f, el proceso se puede ver influenciado por alguna información contextual, llamémosle x, un elemento de un conjunto finito $\mathcal{X}$. En nuestro ejemplo, x podría ser las palabras que rodean a la palabra *in* dentro de la frase en que aparezca.

Nuestra tarea es la de construir un modelo estocástico que fielmente represente el comportamiento del proceso aleatorio. Este modelo será un método de estimar la probabilidad condicional de que dado un contexto x, el proceso obtenga f, o sea $p_{in}(f \mid x)$. Si denotamos con $\mathcal{P}$ el conjunto de todas las distribuciones de probabilidad condicionales, la distribución $p_{in}(f \mid x)$ será un elemento de $\mathcal{P}$.

La muestra de entrenamiento

Para estudiar el proceso, observamos el comportamiento del proceso aleatorio en el tiempo recopilando un gran número de muestras $(x_1, f_1), (x_2, f_2), \cdots, (x_n, f_n)$. En nuestro ejemplo, cada muestra consistirá en un conjunto de palabras x que rodean a *in* dentro de la frase, y la correspondiente traducción f que el proceso obtiene. Por ahora podemos considerar que este conjunto de muestras de entrenamiento han sido generadas por un experto al que se le pasaron aleatoriamente gran cantidad de frases que contenían la palabra *in* y se le preguntó por la mejor traducción en cada caso. Evidentemente en la práctica esto no es lo normal, por lo tanto habrá que ver de qué forma se pueden obtener esta serie de muestras de entrenamiento de una forma automática.

Podemos resumir la muestra de entrenamiento en términos de su distribución de probabilidad empírica $\tilde{p}_{in}$, definida por:

$$\tilde{p}_{in}(x, f) \equiv \frac{1}{N} \times \text{número de veces que } (x, f) \text{ ocurre en la muestra}$$

Típicamente, un par (x, f) en particular no aparecerá en la muestra o a lo sumo aparecerá un número pequeño de veces.

Estadísticos, características y restricciones

Nuestro objetivo es construir un modelo estadístico del proceso que generó la muestra de entrenamiento $\tilde{p}_{in}(x, f)$. La piedra angular del modelo será un conjunto de estadísticos de la muestra de entrenamiento. En nuestro ejemplo hemos empleado bastantes estadísticos: que la frecuencia de que *in* se traduzca por *entre* o *en* sea de $3/10$; que la frecuencia de que *in* se traduzca por *entre* o *a* sea $1/2$; etc. En realidad estos estadísticos son independientes del contexto, pero también podríamos considerar otros que sí dependan de la información que proporciona el contexto x. Por ejemplo, podríamos observar en una muestra de entrenamiento, que si la palabra que sigue a *in* es *April* entonces su traducción es *en* con una frecuencia de $9/19$. Para expresar eventos de este tipo, utilizaremos funciones características (o simplemente características) de la forma:

$$\phi_{in,k}(x, f) = \begin{cases} 1 & \text{if } f = \textit{en} \text{ y } \textit{April} \text{ sigue a } \textit{in} \text{ en } x \\ 0 & \text{en otro caso} \end{cases} . \tag{4.1}$$

donde $k \in 1 \cdots K_{in}$ indica que estamos refiriéndonos a la k-ésima restricción/característica del modelo para la palabra *in*, y donde K_{in} es el número de características relevantes para la palabra *in*.

El valor esperado de $\phi_{in,k}$ con respecto a la distribución empírica $\tilde{p}_{in}(x, f)$ es exactamente el estadístico en el que estamos interesados y se define como:

$$\tilde{p}_{in}(\phi_{in,k}) \equiv \sum_{x,f} \tilde{p}_{in}(x, f) \cdot \phi_{in,k}(x, f) \tag{4.2}$$

Podemos expresar cualquier estadístico que nos interese como el valor esperado de una característica $\phi_{in,k}$ apropiada a cada caso.

Cuando encontramos un estadístico que creemos útil a priori, podemos saber su relevancia/importancia requiriendo a nuestro modelo que dé cuenta de él. Para ello restringimos el valor esperado que el modelo asigna a la correspondiente característica $\phi_{in,k}$. El valor esperado con respecto al modelo $p_{in}(f \mid x)$ es:

$$p_{in}(\phi_{in,k}) \equiv \sum_{x,f} \tilde{p}_{in}(x) \cdot p_{in}(f \mid x) \cdot \phi_{in,k}(x, f) \tag{4.3}$$

donde $\tilde{p}_{in}(x)$ es la distribución empírica de x en la muestra de entrenamiento. Entonces imponemos la restricción de que este valor esperado sea idéntico al valor esperado de $\phi_{in,k}$ en la muestra de entrenamiento. Esto es, requerimos que:

$$p_{in}(\phi_{in,k}) = \tilde{p}_{in}(\phi_{in,k}) \tag{4.4}$$

Combinado las ecuaciones (4.2), (4.3) y (4.4) llegamos a la ecuación:

$$\sum_{x,f} \tilde{p}_{in}(x) \cdot p_{in}(f \mid x) \cdot \phi_{in,k}(x,f) = \sum_{x,f} \tilde{p}_{in}(x,f) \cdot \phi_{in,k}(x,f) \tag{4.5}$$

A la ecuación (4.4) lo denominaremos simplemente "restricción". Restringiendo la atención a aquellos modelos $p_{in}(f \mid x)$ que cumplen la restricción/característica (4.4), estaremos descartando aquellos modelos que no están de acuerdo con la muestra de entrenamiento en cuanto a la frecuencia que el proceso debería mostrar la característica $\phi_{in,k}$.

Resumiendo, ahora tenemos el mecanismo para representar fenómenos estadísticos inherentes a los datos (es decir, $\tilde{p}_{in}(\phi_{in,k})$), y la forma de que nuestro modelo del proceso exhiba estos fenómenos (es decir, $p_{in}(\phi_{in,k}) = \tilde{p}_{in}(\phi_{in,k}), k \in 1 \cdots K_{in}$).

El principio de máxima entropía

Olvidándonos de nuestro ejemplo introductorio, supongamos en general que tenemos K funciones características ϕ_k que determinan los estadísticos que creemos son importantes en el modelado del proceso. Lo que pretendemos es que nuestro modelo esté de acuerdo con esos estadísticos, o lo que es lo mismo, queremos que p esté en el subconjunto $\mathcal{R}$ de $\mathcal{P}$ definido por:

$$\mathcal{R} \equiv \left\{ p \in \mathcal{P} \mid p(\phi_k) = \tilde{p}(\phi_k) \text{ para } k \in \{1, 2, \cdots, K\} \right\} \tag{4.6}$$

La filosofía de máxima entropía nos dice que de todos los modelos $p \in \mathcal{P}$, debemos seleccionar aquel modelo que sea más uniforme. Pero una vez más nos preguntamos cuál es el significado de "uniforme".

Una medida matemática de la uniformidad de una distribución de probabilidad condicional es la entropía condicional, definida del siguiente modo:

$$H(p) \equiv \sum_{x,f} \tilde{p}(x) p(f \mid x) \log p(f \mid x) \tag{4.7}$$

La entropía está acotada inferiormente por cero (la entropía de un modelo con ninguna incertidumbre) y superiormente por $\log \mid \mathcal{F} \mid$ (la entropía de la distribución uniforme de todas los posibles valores de f). Con esta definición en mano, el principio de máxima entropía mantiene que: "Para seleccionar un modelo de un conjunto $\mathcal{R}$ de posibles distribuciones de probabilidad, elegiremos el modelo $p_\star \in \mathcal{R}$ con máxima entropía $H(p)$", es decir:

$$p_\star = \arg\max_{p \in \mathcal{R}} H(p) \tag{4.8}$$

Se puede ver que $p_\star$ está siempre bien definido, es decir, siempre existe un único modelo $p_\star$ con máxima entropía en el conjunto restringido $\mathcal{R}$.

Forma paramétrica

El principio de máxima entropía se nos presenta como un problema de optimización restringida, es decir encontrar el $p_\star \in \mathcal{R}$ que maximice $H(p)$. Para casos simples, como en el ejemplo introductorio con dos restricciones, la solución al problema se puede encontrar de forma analítica. Desafortunadamente, la solución al problema general no se puede escribir explícitamente, con lo que necesitamos una aproximación indirecta.

Para atacar el problema general se utilizan los multiplicadores de Lagrange ampliamente utilizados en la teoría de la optimización restringida. Veamos en que consisten los pasos más relevantes de esta metodología (para más detalles véase [Della Pietra et al. 97]):

- Nos referiremos al problema original de optimización restringida, es decir encontrar $p_\star$ tal que:
$$p_\star = \arg\max_{p \in \mathcal{R}} H(p), \tag{4.9}$$
como el problema *principal*.

- Para cada característica ϕ_i, se introduce un parámetro λ_i (un multiplicador de Lagrange). Se define el Lagrangiano $\Lambda(p, \lambda)$ como
$$\Lambda(p, \lambda) = H(p) + \sum_i \lambda_i (p(\phi_i) - \tilde{p}(\phi_i)) \tag{4.10}$$

- Manteniendo λ fija, se calcula el máximo irrestringido del Lagrangiano $\Lambda(p, \lambda)$ para todo $p \in \mathcal{R}$. Denotamos p_λ el modelo p para el cual $\Lambda(p, \lambda)$ alcanza el máximo siendo $\Psi(\lambda)$ el valor del máximo, es decir:
$$p_\lambda \equiv \arg\max_{p \in \mathcal{R}} \Lambda(p, \lambda) \tag{4.11}$$
$$\Psi(\lambda) \equiv \Lambda(p_\lambda, \lambda) \tag{4.12}$$
A la expresión $\Psi(\lambda)$ se le conoce como la función/problema *dual*. Las funciones p_λ y $\Psi(\lambda)$ se pueden calcular explícitamente como:
$$p_\lambda(f \mid x) = \frac{1}{Z_\lambda(x)} \exp\left(\sum_i \lambda_i \phi_i(x, f)\right) \tag{4.13}$$
$$\Psi(\lambda) = -\sum_x \tilde{p}(x) \log Z_\lambda(x) + \sum_i \lambda_i \tilde{p}(\phi_i) \tag{4.14}$$
donde $Z_\lambda(x)$ es una constante de normalización determinada por el requerimiento de que $\sum_x p_\lambda(f \mid x) = 1$:
$$Z_\lambda(x) = \sum_f \exp\left(\sum_i \lambda_i \phi_i(x, f)\right) \tag{4.15}$$

- Finalmente, el problema *dual* de optimización irrestringida se plantea como: Encontrar $\lambda^\star$ tal que:

$$\lambda^\star = \arg\max_{\lambda} \Psi(\lambda) \tag{4.16}$$

A simple vista no está claro qué es lo que se obtiene con todas estas maquinaciones. De cualquier forma, un principio fundamental en la teoría de los multiplicadores de Lagrange, normalmente conocido como el teorema de Kuhn-Tucker, asegura que bajo ciertas asunciones, el problema *principal* y el *dual* están íntimamente relacionados, resultando en:

Supongamos que $\lambda^\star$ es la solución al problema *dual*, entonces $p_{\lambda^\star}$ es la solución al problema *principal*, esto es $p_{\lambda^\star} = p_\star$. En otras palabras,

> El principio de máxima entropía sujeto a las restricciones $\mathcal{R}$ tiene la forma paramétrica $p_{\lambda^\star}$ de (4.13), donde los parámetros $\lambda^\star$ se pueden determinar maximizando la función *dual* $\Psi(\lambda)$.

La consecuencia práctica más importante de este resultado es que cualquier algoritmo capaz de encontrar el máximo $\lambda^\star$ de $\Psi(\lambda)$ se puede usar para encontrar el máximo $p_\star$ de $H(p)$ para $p \in \mathcal{R}$.

Relación con máxima verosimilitud

El logaritmo de la verosimilitud $L_{\tilde{p}}(p)$ de la distribución empírica $\tilde{p}$ como predicción de un modelo p se define como:

$$L_{\tilde{p}}(p) \equiv \log \prod_{x,f} p(f \mid x)^{\tilde{p}(x,f)} = \sum_{x,f} \tilde{p}(x,f) \log p(f \mid x) \tag{4.17}$$

Es fácil probar que la función dual $\Psi(\lambda)$ de la sección anterior es, de hecho, exactamente el logaritmo de la verosimilitud para un modelo exponencial p_λ, o sea

$$\Psi(\lambda) = L_{\tilde{p}}(p_\lambda) \tag{4.18}$$

Con esta interpretación, el resultado anterior se puede reinterpretar como:

> El modelo $p_\star \in \mathcal{C}$ con máxima entropía es el modelo en forma paramétrica $p_\lambda(f|x)$ que maximiza la verosimilitud de la muestra de entrenamiento $\tilde{p}$.

Este resultado proporciona una justificación añadida al principio de máxima entropía: si la noción de seleccionar un modelo $p_\star$ basándonos en máxima entropía no es lo bastante convincente, también ocurre que este mismo modelo $p_\star$ es también el modelo (de todos los posibles modelos parámetricos de la forma (4.13)) que mejor da cuenta de los datos o la muestra de entrenamiento.

Cálculo de los parámetros

Para todos los problemas, incluso los más simples, encontrar el $\lambda^\star$ que maximice $\Psi(\lambda)$ no se puede hacer de forma analítica con lo que debemos de recurrir a métodos numéricos. Desde el punto de vista de la optimización numérica, la función $\Psi(\lambda)$ se comporta bien dado que es continúa y convexa en λ, de modo que gran variedad de métodos numéricos se pueden utilizar para calcular $\lambda^\star$ como por ejemplo el descenso por gradiente o el gradiente conjugado. Cuando estas técnicas se aplican al problema de máxima entropía se convierten en el popular algoritmo de Brown [Brown 95], aunque existe un método específicamente diseñado para el problema de máxima entropía. Este método es el algoritmo GIS (generalized iterative scaling) [Darroch and Ratcliff 72]. También existe una versión mejorada de este algoritmo, el IIS (improved IS) [Berger et al. 96b].

Nosotros, en todos los experimentos que llevaremos a cabo en esta tesis utilizaremos un toolkit llamado YASMET [Och 00] que implementa el GIS.

4.3 Pros y cons de utilizar ME

La utilización de modelos basados en máxima entropía frente a otros paradigmas de aprendizaje ofrece entre otras las siguientes ventajas:

- Presenta un marco de trabajo en el que es fácil la integración de distintas fuentes de conocimiento.
- No presenta problemas con que varias características se solapen, o sea den cuenta de sucesos equivalente o incluidos en otros.
- Tiene una teoría matemática bien fundada.
- Existen algoritmos eficientes de entrenamiento: GIS, IIS.

Por otra parte también se presentan una serie de desventajas, entre ellas podemos destacar:

- La elección de las características relevantes al problema a modelar pasa por ser todo un arte.
- La extracción de contextos/muestras de entrenamiento requiere un alineamiento entre las frases, e incluso a nivel de palabra dependiendo de la aplicación que le queramos dar.
- No existen métodos eficientes y eficaces de selección automática de característica, siendo además un proceso muy costoso.

4.4 Léxicos mejorados utilizando ME

4.4.1 Motivación

Como hemos visto en el tema anterior, en todos los modelos de IBM, se hace la aproximación $Pr(f_j \mid f_1^{j-1}, a_1^j, e_1^I) = p(f_j \mid e_{a_j})$ eliminando las dependencias con f_1^{j-1}, a_1^{j-1}, y e_1^I (exceptuando e_{a_j}). Evidentemente, esta simplificación no es cierta y demasiado rigurosa para la mayoría de los fenómenos del lenguaje natural. La aproximación más directa para incluir más dependencias en el modelo léxico sería incluir dependencias adicionales como por ejemplo $p(f_j \mid e_{a_j}, e_{a_{j-1}})$, donde se incluyen dependencias de primer orden. Esta aproximación tiene un grave problema, y es el de la falta de datos para poder estimar eficientemente los parámetros del modelo.

Típicamente, los modelos léxicos utilizados en sistemas estadísticos de traducción automática no incluyen ningún tipo de información lingüística o información contextual, lo cual deriva en alineamientos inadecuados entre pares de frases. Por esta razón, definiremos modelos léxicos basados en máxima entropía que sí tengan en cuenta la información contextual; es decir, vamos a crear modelos léxicos dependientes del contexto. En la literatura relacionada con el tema, normalmente a este tipo de modelos se les conoce como modelos desambiguadores. Existen infinidad de trabajos relacionados con la desambiguación léxica, entre ellos podemos destacar a [Ide and Véronis 98] donde podemos encontrar un estado del arte acerca del tema; y ya dentro del marco estadístico tenemos [Brown et al. 91a, Gale et al. 92, Mooney 96, Levow 97, Tomás and Casacuberta 02].

En nuestro caso por tanto el papel que desempeña la máxima entropía es construir modelos estocásticos que eficientemente tengan en cuenta información contextual. Al igual que hemos hecho en la introducción, a partir de ahora, vamos a utilizar $p_e(f \mid x)$ para denotar la probabilidad que el modelo por ME (asociado a la palabra e) asigna a f en el contexto x.

4.4.2 Modelos léxicos por ME

Como hemos comentado anteriormente, en la aproximación por ME describimos las propiedades/características que consideramos relevantes para nuestro problema por medio de funciones características $\phi_{e,k}(x, f), k = 1, \ldots, K_e$ [1].

Por ejemplo, supongamos que la k-ésima característica para la palabra e intenta modelar el fenómeno de la existencia/aparición o ausencia de una palabra específica e'_k en el contexto de la palabra e, la cual puede/debe ser traducida por f'_k.

[1] k indica una característica en concreto de todas las K_e posibles asociadas al modelo por ME para la palabra e.

Podemos expresar esta dependencia usando la siguiente función característica:

$$\phi_{e,k}(x,f) = \begin{cases} 1 & \text{si } f = f'_k \text{ y } e'_k \in x \\ 0 & \text{en otro caso} \end{cases} . \tag{4.19}$$

Consecuentemente, la k-ésima característica para la palabra e tiene asociado el par (e'_k, f'_k) en este caso.

Como hemos visto en el apartado anterior, el principio de ME indica que el modelo $p_e(f \mid x)$ tiene una forma paramétrica óptima teniendo en cuenta las funciones características $\phi_{e,k}, k = 1, \ldots, K_e$, y viene dada por la expresión:

$$p_e(f \mid x) = \frac{1}{Z_{\Lambda_e}(x)} \exp\left(\sum_{k=1}^{K_e} \lambda_{e,k}\phi_{e,k}(x,f)\right) . \tag{4.20}$$

El modelo resultante tiene una forma exponencial con $\Lambda_e \equiv \{\lambda_{e,k}, k = 1, \ldots, K_e\}$ parámetros libres. Por lo tanto, la estimación de los valores de estos parámetros por máxima verosimilitud, es decir que maximizan la verosimilitud de un corpus dado de entrenamiento, se puede calcular utilizando el algoritmo GIS (como ya hemos visto anteriormente).

Es importante notar que, en principio, obtendremos tantos modelos por ME de este tipo como palabras haya en el vocabulario de destino $\mathcal{E}$. De todas formas para evitar problemas de falta de datos para aquellas palabras poco vistas en el conjunto de entrenamiento (palabras raras) crearemos modelos por ME solamente para aquellas palabras que aparezcan un número determinado de veces en nuestro conjunto de entrenamiento.

4.4.3 Información contextual y eventos de entrenamiento

Para entrenar un modelo por ME $p_e(f \mid x)$ asociado a una palabra destino e, necesitaremos construir, a partir de un corpus bilingüe, la correspondiente muestra de entrenamiento dependiendo de la información contextual que queramos utilizar.

Para el caso que nos compete, es decir construir modelos léxicos dependientes del contexto, necesitamos saber de alguna forma cuáles son las palabras f candidatas a ser estimadas por el modelo, es decir las posibles traducciones f de una palabra e en concreto. En principio los posibles valores de f serán cualquier palabra del vocabulario $\mathcal{F}$, siendo el contexto a tener en cuenta, el contexto en el que ese par (e,f) concreto aparece dentro de todas los pares de frases del corpus de entrenamiento. Por lo tanto, para nosotros un *evento*/muestra de entrenamiento, llamémosle (f,e,x), estará formado por tres elementos:

- La palabra e para la cual construir el modelo por ME,
- La correspondiente palabra origen f, traducción de e en el contexto x.

- El contexto x en el cual el par (f, e) aparece. Donde x variará dependiendo del contexto que se elija para el par específico (e, f).

El número de ocurrencias de un evento en la muestra de entrenamiento también se considera como parte de dicho evento. A este número le llamaremos *contador* del evento de entrenamiento.

Una vez definidos los eventos de entrenamiento, necesitamos definir la información contextual a extraer. Nosotros usaremos como información contextual la información que nos proporcionan las palabras que rodean al par origen-destino (f, e) alineado. Más concretamente, utilizaremos la siguiente información contextual:

- Contexto destino: Al igual que en [Berger et al. 96b] consideramos una ventana de δ palabras a la izquierda y a la derecha de la palabra destino e considerada. Con este contexto se intenta capturar algunos fenómenos lingüísticos como lo son los pares articulo-nombre o el típico intercambio nombre-adjetivo a adjetivo-nombre entre los idiomas origen y destino.

- Contexto origen: Adicionalmente, consideramos una ventana [2] de δ palabras a izquierda de la palabra origen f, la cual es considerada la traducción de e. Obviamente, la palabra origen f también se considera. Con esta información podemos tener en cuenta la información adicional que nos proporciona el modelo de lenguaje, como por ejemplo bigramas de historia variable o los habitualmente conocidos como *trigger pairs* [Tillmann and Ney 96, Tillmann and Ney 97, Zhou and Lua 98].

- Clases de palabras: además de usar dependencias con la identidad de las palabras también utilizamos dependencias con las clases a las que pertenecen las palabras del contexto. Haciendo esto mejoramos la generalización de los modelos e incluimos de alguna manera información sintáctica y semántica de las palabras pertenecientes al contexto. Las clases de palabras se obtendrán de forma automática utilizando el método estadístico de clustering propuesto en [Och 99] que normalmente obtiene clases de palabras con el mismo significado semántico.

Dependiendo de la utilidad que le queramos dar a nuestros modelos por ME podremos delimitar el conjunto de eventos de entrenamiento a aquellos que a priori sean más significativos/discriminativos. Por ejemplo si nuestros modelos basados en ME los queremos dedicar a construir un modelo desambiguador podríamos partir de un corpus previamente alineado de modo que el conjunto posible de palabras f a ser estimadas para cada palabra e se verá considerablemente reducido a aquellos pares de palabras que realmente aparecen alineados en nuestro corpus, es decir, a aquellas posibles traducciones a priori más prometedoras. Obtener un corpus alineado es bastante fácil con lo que ya conocemos,

[2] En principio el δ para los contextos destino y origen no tiene porque ser iguale pero por claridad en la notación lo consideramos como tal.

solamente deberíamos lanzar un entrenamiento (para cualquiera de los modelos de IBM convencionales) y calcular el alineamiento por Viterbi para cada par de frases del conjunto de entrenamiento. Este aspecto lo comentaremos con más detalle cuando hablemos de la integración de los modelos de ME en el entrenamiento de los modelos convencionales. En definitiva, para construir modelos desambiguadores para las palabras a priori o hipotéticamente ambiguas procederemos del siguiente modo:

1. Encontrar el alineamiento por Viterbi del corpus de entrenamiento para un modelo de IBM en concreto.
2. Encontrar la traducción del conjunto de test.
3. Extraer del corpus alineado los eventos de entrenamiento (f, e, x) para toda posible palabra ambigua e.
4. Realizar un entrenamiento de cada modelo $p_e(f \mid x)$ utilizando el algoritmo GIS.
5. Retraducir (como fase de post-proceso) la traducción de las frases de test utilizando los modelos por ME como modelos desambiguadores.

Para construir esta muestra, necesitamos primero conocer el alineamiento a nivel de palabra entre cada par de frases en el corpus. Como ya hemos comentado este alineamiento se obtiene utilizando el alineamiento por Viterbi que nos proporcione el modelo de traducción que estemos utilizando. Concretamente, nosotros utilizaremos el alineamiento por Viterbi para el modelo 5.

4.4.4 Definición de características

Para definir las características que consideramos relevantes a los modelos léxicos dependientes del contexto que vamos a considerar se pueden definir mediante una tripleta (pos, et_1, et_2), donde:

- pos: es la posición que et_2 ocupa en el contexto.
- et_1: será la palabra f a predecir o la clase a la que pertenece, o sea $\mathcal{F}_c(f)$.
- et_2: será cualquiera de las palabras e o f o sus respectivas clases, o sea $\mathcal{E}_c(e)$ o $\mathcal{F}_c(f)$.

Usando esta notación y dado un contexto definido como:

$$x = \boxed{e_{i-\delta} \ldots e_i \ldots e_{i+\delta}} \, \boxed{f_{j-\delta} \ldots f_j}$$

para el par alineado (f_j, e_i) tendremos las siguientes categorías de características:

1. $(0, f_j,)$

2. $(\pm 1, f_j, e')$ y $e' = e_{i\pm 1}$
3. $(\pm \delta, f_j, e')$ y $e' \in \{e_{i-\delta} ... e_{i+\delta}\}$
4. $(\pm 1, \mathcal{F}_c(f_j), \mathcal{E}_c(e'))$ y $e' = e_{i\pm 1}$
5. $(\pm \delta, \mathcal{F}_c(f_j), \mathcal{E}_c(e'))$ y $e' \in \{e_{i-\delta} ... e_{i+\delta}\}$
6. $(-1, f_j, f')$ y $f' = f_{j-1}$
7. $(-\delta, f_j, f')$ y $f' \in \{f_{j-\delta} ... f_{j-1}\}$
8. $(-1, \mathcal{F}_c(f_j), \mathcal{F}_c(f'))$ y $f' = f_{j-1}$
9. $(-\delta, \mathcal{F}_c(f_j), \mathcal{F}_c(f'))$ y $f' \in \{f_{j-\delta} .. f_{j-1}\}$

La categoría 1 da cuenta de las restricciones que aseguran la igualdad entre la probabilidad de la traducción de cualquier f_j de e_i de acuerdo al modelo y la probabilidad de esa traducción en la distribución empírica. Un modelo por ME que sólo tenga en cuenta características de esta categoría 1 predecirá cada posible traducción f_j con probabilidad $\tilde{p}_{e_i}(f_j)$ de acuerdo a los datos de entrenamiento. Ésta es exactamente la empleada en los modelos de traducción de IBM [Brown et al. 93], es decir $\tilde{p}_{e_i}(f_j) \equiv t(f_j \mid e_i)$. La categoría 2 describe características para el par (f_j, e_i) en caso en que otra palabra específica e' sea exactamente la palabra que ocupa la posición justo a la izquierda o justo a la derecha de e_i en el contexto x. El mismo comentario es válido para la categoría 3 sólo que en este caso la palabra e' puede ocupar cualquier posición $(-\delta \cdots + \delta)$ en el contexto x. Las categorías 4 y 5 son análogas a las 2 y 3 pero utilizando las clases de las palabras en vez de la identidad de la palabras. Los mismos comentarios son válidos para las categorías 6, 7, 8 y 9 pero con respecto a la parte del contexto origen en vez del contexto destino. Una idea más intuitiva acerca de las diferentes categorías la podemos encontrar en la Tabla 4.1.

4.4.5 Selección de características

Como todo proceso de aprendizaje basado en un conjunto de características/restricciones a cumplir por el modelo a aprender es necesario realizar una selección de aquellas características que a priori sean relevantes al proceso a modelar.

En nuestro caso esto es especialmente importante, pues como podemos ver en la Tabla 4.1 el número de características (número de parámetros a estimar) para cada modelo por ME es $(8 \cdot \mid \mathcal{E} \mid \cdot \mid \mathcal{F} \mid) + \mid \mathcal{E} \mid$.

La selección de características la llevaremos a cabo atendiendo a estos dos criterios:

1. Construiremos un modelo por ME solamente para aquellas palabras e que aparezcan en el corpus de entrenamiento más de un número N de veces.

Categoría	# de características reales	$\phi_{e_i,k}(x, f_j) = 1$ si y sólo si ...
1	$\mid \mathcal{E} \mid$	$f_j = \diamond$
2	$\mid \mathcal{E} \mid \cdot \mid \mathcal{F} \mid$	$f_j = \diamond$ and $\square \in$ $\square$ $\square$ $\bullet$ e_i $\square$ $\square$ $\square$
2	$\mid \mathcal{E} \mid \cdot \mid \mathcal{F} \mid$	$f_j = \diamond$ and $\square \in$ $\square$ $\square$ $\square$ e_i $\bullet$ $\square$ $\square$
3	$\mid \mathcal{E} \mid \cdot \mid \mathcal{F} \mid$	$f_j = \diamond$ and $\square \in$ $\bullet$ $\bullet$ $\bullet$ e_i $\square$ $\square$ $\square$
3	$\mid \mathcal{E} \mid \cdot \mid \mathcal{F} \mid$	$f_j = \diamond$ and $\square \in$ $\square$ $\square$ $\square$ e_i $\bullet$ $\bullet$ $\bullet$
6	$\mid \mathcal{E} \mid \cdot \mid \mathcal{F} \mid$	$f_j = \diamond$ and $\diamond \in$ $\square$ $\square$ $\bullet$ f_j
7	$\mid \mathcal{E} \mid \cdot \mid \mathcal{F} \mid$	$f_j = \diamond$ and $\diamond \in$ $\bullet$ $\bullet$ $\bullet$ f_j

Tabla 4.1: Significado de las diferentes categorías de características donde $\square$ representa un palabra destino en concreto (a ocupar la posición $\bullet$) y $\diamond$ representa una palabra origen en concreto, donde k tiene asociado el par ($\square$,$\diamond$).

2. Para reducir el número de características realizamos una selección basado en conteo, de modo que aquellas características que no aparezcan más de un número T de veces serán descartadas.

El objetivo final de la selección de características es doble, por un lado obtenemos modelos menores utilizando menor número de características y por otro evitar el sobre-entrenamiento de dichos modelos.

4.5 Estudio experimental

Llegados a este punto ya somos capaces de utilizar los modelos basados en ME definidos en la sección anterior dentro del marco de un sistema de traducción automática estadística.

Teniendo en cuenta la definición de los modelos léxicos dependientes del contexto que acabamos de comentar cabría hacernos la siguiente pregunta: ¿Cómo se pueden utilizar los modelos basados en ME para mejorar sistemas estadísticos de traducción automática?

En esta sección vamos a responder a esta pregunta de forma experimental y para ello realizaremos dos tipos de experimentos distintos:

- El primero será llevar a cabo una comparativa de los modelos obtenidos teniendo y sin tener en cuenta la ME. Esto lo realizaremos mediante un experimento de evaluación de la perplejidad de dichos modelos.

- El segundo será un experimento de traducción utilizando los modelos basados en ME en una fase de postproceso a la salida de un sistema de traducción estadística. Esto lo llevaremos a cabo mediante un algoritmo de repuntuación de hipótesis.

Por otra parte, estos experimentos nos servirán también para hacernos una idea de que tipo de información contextual es útil y para establecer un valor razonable del parámetro T para futuros experimentos.

Como hemos comentado en la sección anterior, el punto de partida para poder hacer uso de los modelos léxicos basados en ME dentro del marco de un sistema estadístico de traducción es disponer de un corpus alineado (a nivel de palabra) para poder llevar a cabo todos los aspectos concernientes al entrenamiento de la serie específica de modelos por ME a considerar. De este modo nosotros haremos uso del alineamiento por Viterbi para el modelo 5 que proporciona el programa GIZA++[Och 00].

Para realizar los experimentos que se muestran a continuación en esta sección hemos utilizado la tarea de VERBMOBIL (ver sección 2.7.2, página 46).

4.5.1 Evaluación de la perplejidad de los modelos

Como comentamos en el capítulo 2 la perplejidad de un modelo de traducción nos da una idea de cuan bueno es un modelo con respecto a un conjunto de datos de test.

Para establecer de forma ideal los parámetros a utilizar en un experimento de traducción parece razonable utilizar un conjunto de validación y estudiar su perplejidad. Para ello hemos dividido el conjunto de entrenamiento mostrado en la tabla 2.3 en un conjunto nuevo de entrenamiento y de validación para la tarea VERBMOBIL, dado que este corpus fue el que se utilizó en un sistema de traducción previamente desarrollado y que a continuación utilizaremos para presentar algunos experimentos de repuntuación de hipótesis. Las estadísticas de este nuevo conjunto se muestran en la tabla 4.2.

		Alemán	Inglés
Entrenamiento	Frases	50 000	
	Palabras	519 523	549 521
	Vocabulario	7 940	4 673
Validación	Frases	8073	
	Palabras	64 875	65 547
	Vocabulario	2 579	1 666
Test	Frases	147	
	Palabras	1 968	2 173
	PP (ML trigrama)	(40.3)	28.8

Tabla 4.2: Características del corpus empleado en los experimentos de calidad de la perplejidad y en los experimentos de repuntuación de hipótesis.

Los resultados con respecto a la perplejidad de los conjuntos de entrenamiento y validación se muestran en la tabla 4.3 para distintos valores del parámetro T

que define el número de características a tener en cuenta en el entrenamiento. Es decir aquellas características que aparecen un número menor de T en los eventos de entrenamiento de los modelos por ME serán descartadas; por lo tanto un valor de $T = 0$ indica que todas las características vistas serán utilizadas en el entrenamiento de los modelos, mientras que un valor de $T = 32$ indica que aquellas características vistas menos de 32 veces serán descartadas. En la misma tabla se presentan resultados teniendo en cuenta diferente información contextual, teniendo en cuenta la información del inglés y teniendo en cuenta la información de ambos idiomas. En el experimento el valor de T se eligió que variara en un rango de 0 a 512, incrementando en potencias de 2. Por otro lado se seleccionaron aquellas palabras inglesas que aparecen en el corpus de entrenamiento un número mayor de 150 (es decir $N = 150$), lo cual redunda en la elección de 348 palabras de las 4673 posibles. De este modo se han construido 348 modelos por ME para las palabras seleccionadas. A título de ejemplo en la tabla 4.4 se muestran el número de características empleadas de acuerdo a diferentes valores de T para las 348 palabras elegidas.

Hay que matizar que estos experimentos se realizaron con un entrenamiento por máxima entropía de forma desacoplada, utilizando la información del alineamiento por Viterbi para el modelo 5 de un entrenamiento convencional del corpus, siguiendo el mismo esquema de traducción previamente utilizado.

	Inglés		Inglés+Alemán	
T	TrainPP	ValiPP	TrainPP	ValiPP
0	5.03	11.39	4.60	9.28
2	6.59	10.37	5.70	8.94
4	7.09	10.28	6.17	8.92
8	7.50	10.39	6.63	9.03
16	7.95	10.64	7.07	9.30
32	8.38	11.04	7.55	9.73
64	9.68	11.56	8.05	10.26
128	9.31	12.09	8.61	10.94
256	9.70	12.62	9.20	11.80
512	10.07	13.12	9.69	12.45

Tabla 4.3: Perplejidades de los conjuntos de entrenamiento y validación utilizando distinta información contextual (izquierda inglés, derecha inglés+alemán+categorías) y diferentes umbrales de corte del parámetro T. Las perplejidades de referencia obtenidas en la última iteración del modelo 5 fueron respectivamente TrainPP = 10.38 y ValiPP = 13.22.

Como era de esperar la reducción de la perplejidad del conjunto de validación es menor que la del conjunto de entrenamiento, pero en ambos casos, la perplejidad se reduce cuando se utilizan los modelos basados en ME. Este resultado se corrobora a la vista de las perplejidades mostradas en las tablas 4.10 y 4.11.

# características usadas		
T	Inglés	Inglés+Alemán
0	846121	1581529
2	240053	500285
4	153225	330077
8	96983	210795
16	61329	131323
32	40441	80769
64	28147	49509
128	21469	31805
256	18511	22947
512	17193	19027

Tabla 4.4: Número de características usadas de acuerdo a diferentes umbrales de corte T, utilizando distinta información contextual.

4.5.2 Experimento de repuntuación de hipótesis

Como primera aplicación de los modelos contextuales en la traducción hemos implementado un algoritmo de repuntuación de hipótesis basado en estos modelos (ver algoritmo 4.1). Este algoritmo toma como entrada el modelo léxico convencional $t(f \mid e)$ y los 348 modelos léxicos obtenidos con el entrenamiento por ME $p_e(f \mid x)$, de modo que para una hipótesis de entrada e_1^I y el correspondiente alineamiento el algoritmo modifica el score $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$ teniendo en cuenta la información que le proporcionan los modelos léxicos refinados. Dependiendo de la ambigüedad en la traducción de las palabras que conformen la hipótesis de entrada, el score aumentará o disminuirá consecuentemente. El algoritmo se aplica reiteradamente al conjunto de las N-mejores traducciones obtenidas con un traductor propuesto en [Tillmann 01], el cual implementa un algoritmo de decodificación para el modelo 4 basado en técnicas de programación dinámica. En realidad dicho traductor es subóptimo con lo que las N-mejores (10 en nuestro caso) traducciones no lo son en realidad. De cualquier forma este experimento preliminar nos da una idea de cómo puede afectar la inclusión de los modelos léxicos basados en ME dentro de un sistema de traducción automática estadística.

En la tabla 4.5 se muestran los resultados de traducción en términos de WER y PER para diferentes valores del umbral T y para distinta información contextual utilizando el conjunto de test descrito en la tabla 4.2. Como podemos ver en estos resultados, en general la calidad de la traducción mejora pero muy levemente, si la comparamos con los resultados obtenidos a nivel de perplejidades. Este pequeño incremento en gran parte es debido a la no optimalidad del conjunto de las N-mejores traducciones obtenidas con el traductor comentado.

Algoritmo: ***ME_rescoring***
entrada: $\mathbf{f}$, $\mathbf{a}$, $\{\mathbf{e}_n : nbest\_hip(\mathbf{f}), 1 \leq n \leq N\}$, $t(f \mid e)$, $\{p_e(f \mid x) : \forall_e \in ME\}$
salida: $\mathbf{e} = \arg\max_{\mathbf{e}_n \in nbest\_hip(\mathbf{f})} p_\Lambda(\mathbf{f}, \mathbf{a} \mid \mathbf{e}_n)$
  para_todo $\mathbf{e}_n \in nbest\_hip(\mathbf{f})$ hacer
    $\hat{\mathbf{e}} = \mathbf{e}_n$
    $p_\Lambda(\mathbf{f}, \mathbf{a} \mid \mathbf{e}_n) = p_\theta(\mathbf{f}, \mathbf{a} \mid \mathbf{e}_n)$
    para $j = 1 \cdots J$ hacer
      $p_\Lambda(\mathbf{f}, \mathbf{a} \mid \mathbf{e}_n) = p_\Lambda(\mathbf{f}, \mathbf{a} \mid \mathbf{e}_n) \cdot \frac{p_e(f_j \mid x_{a_j})}{t(f_j \mid e_{n\,a_j})}$
    fin_para
    si $(p_\Lambda(\mathbf{f}, \mathbf{a} \mid \mathbf{e}_n) > p_\theta(\mathbf{f}, \mathbf{a} \mid \mathbf{e}_n))$ entonces
      $\hat{\mathbf{e}} = \mathbf{e}_n$
    fin_si
  fin_para
  devolver: $\hat{\mathbf{e}}$

Algoritmo 4.1: Algoritmo de repuntuación de hipótesis basado en ME

En el futuro planeamos realizar esta misma experimentación con conjuntos de N-mejores traducciones reales o en su defecto con grafos de traducción.

	Inglés		Inglés+Alemán	
T	WER	PER	WER	PER
0	54.57	42.98	54.02	42.48
2	54.16	42.43	54.07	42.71
4	54.53	42.71	54.11	42.75
8	54.76	43.21	54.39	43.07
16	54.76	43.53	54.02	42.75
32	54.80	43.12	54.53	42.94
64	54.21	42.89	54.53	42.89
128	54.57	42.98	54.67	43.12
256	54.99	43.12	54.57	42.89
512	55.08	43.30	54.85	43.21

Tabla 4.5: Resultados de traducción tras ejecutar el algoritmo de repuntuación a una lista de las 10 mejores traducciones para 147 frases de test de la tarea VERBMOBIL. Los resultados se muestran para diferentes valores de T e información contextual. Los resultados base para este experimento fueron de un WER=54.80 y un PER=43.07.

En la tabla 4.6 podemos ver algunos ejemplos reales en los que la traducción obtenida tras el proceso de repuntuación obtiene mejores resultados que la mejor traducción obtenida con el traductor utilizado.

SRC:	Danach wollten wir eigentlich noch Abendessen gehen.
M4:	We actually concluding dinner together.
ME:	Afterwards we wanted to go to dinner.
SRC:	Bei mir oder bei Ihnen?
M4:	For me or for you?
ME:	At your or my place?
SRC:	Das wäre genau das richtige.
M4:	That is exactly it spirit.
ME:	That is the right thing.
SRC:	Ja, das sieht bei mir eigentlich im Januar ziemlich gut aus.
M4:	Yes, that does not suit me in January looks pretty good.
ME:	Yes, that looks pretty good for me actually in January.

Tabla 4.6: Algunos ejemplos de traducciones obtenidas utilizando los modelos por ME y el algoritmo de repuntuación para una frase de entrada en alemán.

4.6 Inclusión de modelos léxicos por ME en el aprendizaje de los modelos de IBM

Una gran ventaja de los modelos de ME es que se pueden incluir en el aprendizaje de los modelos convencionales de traducción. En esta sección veremos con todo detalle como realizar esta integración, y lo que es más, cómo realizarla de forma eficiente dado el gran coste computacional que conlleva.

4.6.1 Integración en el entrenamiento

Antes de nada vamos a recordar cómo se realiza el entrenamiento de los parámetros de los modelos convencionales. Cada modelo tiene un conjunto específico de parámetros libres θ; por ejemplo el modelo 4 consiste en parámetros léxicos, de alineamiento y de fertilidades, es decir:

$$\theta = \left\{ \{t(f \mid e)\}, \{d_{=1}(\Delta j)\}, \{d_{>1}(\Delta j)\}, \{n(\phi \mid e)\}, p_1 \right\} \ . \tag{4.21}$$

Para entrenar los parámetros θ del modelo llevamos a cabo un estimación por máxima verosimilitud utilizando un corpus paralelo consistente en S pares de frases $\{(\mathbf{f}_s, \mathbf{e}_s) : s = 1, \ldots, S\}$, de modo que el valor óptimo de los parámetros vendrá dado una vez más por:

$$\hat{\theta} = \arg\max_{\theta} \prod_{s=1}^{S} \sum_{\mathbf{a}} p_\theta(\mathbf{f}_s, \mathbf{a} \mid \mathbf{e}_s) \ . \tag{4.22}$$

Como ya hemos comentado en el tema anterior, esta estimación la hacemos mediante el algoritmo EM [Baum 72, Dempster et al. 77]. En el paso de estimación

(paso-E), los contadores para los parámetros léxicos para un par de frases $(\mathbf{e}, \mathbf{f})$ se calculan del siguiente modo:

$$c(f \mid e; \mathbf{e}, \mathbf{f}) = N(\mathbf{e}, \mathbf{f}) \cdot \sum_{\mathbf{a}} Pr(\mathbf{a} \mid \mathbf{e}, \mathbf{f}) \sum_{j} \delta(f, f_j)\delta(e, e_{a_j}) \ . \tag{4.23}$$

Aquí, $N(\mathbf{e}, \mathbf{f})$ es el número de veces que el correspondiente par aparece en el corpus, que normalmente valdrá 0 ó 1.

En el paso de maximización (paso-M), queremos calcular los parámetros léxicos $\hat{t}(f \mid e)$ que maximizan la verosimilitud de corpus de entrenamiento. Esto resulta en la siguiente reestimación [Brown et al. 93]:

$$t(f \mid e) = \frac{\sum_s c(f \mid e; \mathbf{f}^{(s)}, \mathbf{e}^{(s)})}{\sum_{s,f} c(f \mid e; \mathbf{f}^{(s)}, \mathbf{e}^{(s)})} \ . \tag{4.24}$$

Las probabilidades pertenecientes a los modelos de alineamiento y fertilidad se pueden calcular del mismo modo.

En el caso de los modelos por ME, para una palabra destino e en concreto, debemos encontrar los parámetros definidos por $\Lambda_e \equiv \{\lambda_{e,k} : k = 1, \ldots, K_e\}$ en vez de los parámetros $\{t(f \mid e)\}$. En este caso seguimos la siguiente aproximación:

En el paso-E realizamos un recolección refinada de contadores para los parámetros léxicos dependientes del contexto:

$$c(f \mid e, x; \mathbf{e}, \mathbf{f}) = N(\mathbf{e}, \mathbf{f}) \cdot \sum_{\mathbf{a}} Pr(\mathbf{a} \mid \mathbf{e}, \mathbf{f}) \sum_{j} \delta(f, f_j)\delta(e, e_{a_j})\delta(x, x_{j,a_j}) \ . \tag{4.25}$$

Donde x_{j,a_j} denota el contexto que rodea al par alineado (f_j, e_{a_j}).

En el paso-M, calculamos los parámetros léxicos dependientes del contexto que maximizan la verosimilitud de la muestra de entrenamiento, es decir:

$$\hat{\Lambda}_e = \arg\max_{\Lambda_e} \prod_{f,x} c(f \mid e, x; \mathbf{e}, \mathbf{f}) \cdot \log p_e(f \mid x) \ . \tag{4.26}$$

De modo que los contadores léxicos refinados $c(f \mid e, x; \mathbf{e}, \mathbf{f})$ serán los pesos del conjunto de eventos de entrenamiento (f, e, x), los cuales se utilizarán para entrenar los modelos por ME.

Al igual que en los modelos IBM convencionales, la reestimación de las probabilidades de alineamiento y de fertilidad no cambian si utilizamos los modelos léxicos por ME.

Por lo tanto, para este caso cada iteración del algoritmo EM quedará de la siguiente forma:

1. paso-E:
   (a) Recolectar contadores para los parámetros de alineamiento y fertilidad.

(b) Recolectar contadores refinados para los parámetros léxicos dependientes del contexto y generar los eventos de entrenamiento para los modelos por ME.

2. paso-M:

(a) Reestimar los parámetros de alineamiento y fertilidad.

(b) Realizar un entrenamiento con el GIS para los parámetros léxicos dependientes del contexto.

Con respecto al bucle EM convencional, en este caso los pasos 1b y 2b conllevan un incremento en espacio y tiempo de computación. En la siguiente sección veremos cómo atacar este problema de modo que la integración de los modelos por ME en el algoritmo de entrenamiento convencional sea lo más eficiente posible.

4.6.2 Entrenamiento eficiente

Como introducción a la integración del entrenamiento por ME dentro del algoritmo EM vamos a suponer que nos encontramos en la iteración k-ésima del algoritmo EM. En el paso-E de esta iteración, para cada par de frases (e_1^I, f_1^J) en el corpus de entrenamiento y para cada posible alineamiento entre ellas a_1^J, necesitamos usar las probabilidades del modelo léxico por ME calculadas en la iteración $k-1$. Por lo tanto necesitaremos, a priori, recalcular $p_{e_i}(f_j \mid x)$ para cada cálculo de $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$ (y todo ello J veces para cada una de las J palabras de f_1^J). Para realizar este cálculo eficientemente, precalcularemos todas las posibles $p_{e_i}(f_j \mid x)$ en una matriz $(I \times J)$ de probabilidades léxicas de traducción por ME. Por lo tanto, cada vez que necesitemos calcular esa probabilidad sólo necesitaremos acceder al correspondiente elemento de la matriz. En el paso-E también se realiza la correspondiente generación de los eventos de entrenamiento (f_j, e_i, x) de la iteración actual.

Una vez llevado a cabo el paso-E para cada par de frases en el corpus, tendremos todos los posibles eventos (f, e, x) para cada palabra e. Entonces, con estos eventos, en el paso-M realizaremos un entrenamiento mediante el algoritmo GIS del modelo por ME asociado a cada palabra e que consideremos a priori relevante para nuestro problema y, por tanto el conjunto de parámetros Λ_e que define a cada modelo por ME específico.

Especialmente, los factores que influyen en el incremento computacional que conlleva esta integración son los siguientes:

1. El tiempo de computación requerido para construir la matriz de traducción. Esto conllevará un incremento de un orden de magnitud.

2. El tiempo de computación del algoritmo GIS para entrenar el modelo asociado a cada palabra e a considerar (en el peor de los casos cuando todas

las palabras $e \in \mathcal{E}$ se tengan en cuenta). En los experimentos que hemos llevado a cabo, el tiempo de computación del algoritmo GIS varía en media de 5 a 10 segundos. Esto puede provocar un incremento en el tiempo de computación de hasta 2 órdenes de magnitud dependiendo del número de modelos por ME a considerar.

Por lo tanto, el tiempo de computación adicional requerido por la integración depende directamente en el número de modelos por ME a considerar. Como hemos comentado en la sección anterior, solamente desarrollaremos modelos por ME para aquellas palabras que aparezcan más de un número determinado de veces. En los experimentos llevados a cabo, esta selección deriva en utilizar solamente del 5 al 10% del tamaño del vocabulario.

Con respecto al incremento espacial, necesitaremos almacenar cada posible evento de entrenamiento, es decir cada posible combinación de $e \in \mathcal{E}, f \in \mathcal{F}$ y x. Obviamente esto requiere una cantidad enorme de memoria, por lo tanto la selección de palabras para las cuales construir un modelo por ME es de especial importancia también para la eficiencia espacial.

Por otra parte, el número de eventos de entrenamiento es un factor importante en el coste computacional del algoritmo GIS. Por este motivo proponemos también una poda a nivel de eventos de entrenamiento. Como se describe al final de la sección anterior, los contadores léxicos refinados (contadores fraccionales [Brown et al. 93]) serán los pesos de los eventos de entrenamiento, por lo tanto eliminaremos aquellos eventos cuyos contadores fraccionales sean menores que 1. Esto es, los eventos raros (o prácticamente no vistos en training) se descartan y por lo tanto el entrenamiento GIS será más rápido y la estimación de los parámetros de los modelos por ME será mejor y más precisa. Del mismo modo esta poda conlleva una reducción en los requerimientos espaciales del algoritmo.

En lo que resta de la sección proponemos una simplificación a la aproximación anterior, la cual reduce sustancialmente el incremento computacional que la integración de los modelos por ME conlleva.

Esta simplificación consiste en: Primero realizar un entrenamiento normal del algoritmo EM para los modelos de IBM, y entonces, después de la última iteración del algoritmo realizar un entrenamiento de los modelos por ME solamente una vez utilizando únicamente el alineamiento por Viterbi para cada par de frases de entrenamiento. Finalmente se realiza un nuevo entrenamiento EM fijando los parámetros léxicos a los obtenidos mediante el entrenamiento por máxima entropía. En este caso la información contextual que proporcionan los modelos por ME también se tiene en cuenta pero de forma desacoplada, por contra de la aproximación anterior, la cual es completamente integrada.

Es importante destacar que en esta aproximación solamente se necesita un entrenamiento por ME. Además, en vista de los resultados obtenidos, la calidad de los alineamientos siguiendo esta aproximación y la aproximación completamente integrada son prácticamente los mismos.

En la Tabla 4.7, podemos ver el tiempo medio consumido (en segundos en una

máquina Pentium III 600 MHz) por iteración del algoritmo EM de las diferentes aproximaciones para dos tareas distintas VERBMOBIL y HANSARDS. En este experimento, no se consideró la información contextual referente al contexto destino por obtener los mismos resultados que con la información contextual completa.

Tarea	Tamaño de cjto. entrenamiento	# de e	Aproximación		
			conv.	ME	ME Simpl.
Verbmobil	0.5K	29	1	29	1.5
	8K	84	18	235	68
	35K	209	60	2290	675
Hansards	0.5K	15	2.5	29	3
	8K	80	35	1180	100
	128K	1214	655	16890	6870

Tabla 4.7: Tiempo medio de computación por iteración de las diferentes aproximaciones (convencional, ME y ME simplificada).# de e es el número de palabras destino a ser modeladas por ME después de la selección basada en conteo.

4.7 Evaluación de la calidad de los alineamientos

4.7.1 Metodología de evaluación

Para evaluar la calidad de los alineamientos obtenidos con la técnica expuesta en este capítulo hemos utilizado una metodología de evaluación propuesta en [Och and Ney 00a] y utilizada en otros trabajos como [Och and Ney 03, García-Varea et al. 02b, García-Varea et al. 02a]. Esta metodología se basa en un esquema de anotación creado manualmente por expertos lingüistas que explícitamente permite alineamientos ambiguos a nivel de palabra. A los expertos que construyen el esquema de anotación se les pregunta por dos tipos diferentes de alineamientos: alineamientos $S(eguros)$, los cuales se utilizan para especificar alineamientos no ambiguos, y alineamientos $P(osibles)$ utilizados para especificar alineamientos ambiguos. La etiqueta P se utiliza en particular para alinear palabras con expresiones idiomáticas, traducciones libres y palabras función (conjunciones, posesivos, preposiciones y pronombres) ausentes. Evidentemente siempre se cumplirá que $(S \subseteq P)$.

Por lo tanto los alineamientos de referencia podrán contener relaciones muchos a uno y uno a muchos. En la figura 4.1 podemos ver dos ejemplos (de la tarea HANSARDS de frases alineadas manualmente con ambos tipos (S y P) de etiquetas.

La calidad de un alineamiento $A = \{(j, a_j) \mid a_j > 0\}$ se calcula mediante las

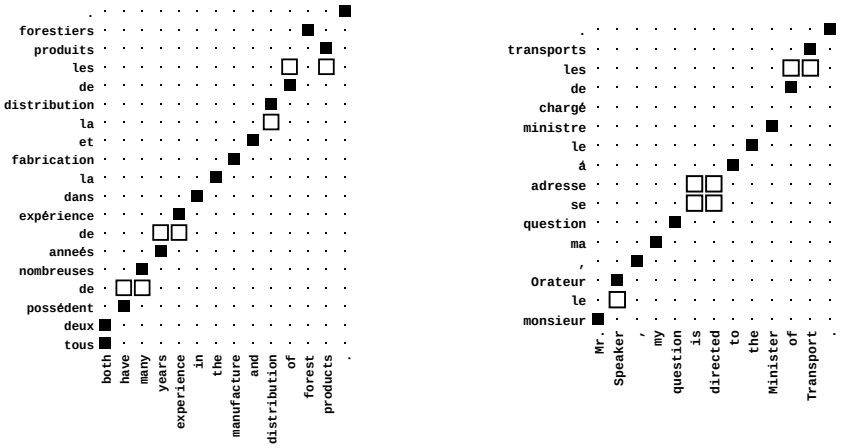


Figura 4.1: Dos ejemplos de alineamiento manual con alineamientos $S(eguros)$ (■) y $P(osibles)$ (□).

medidas *precision* y *recall* [Manning and Schütze 01], convenientemente redefinidas para el caso que nos compete, y mediante la ratio de errores de alineamientos (AER[3]), el cual se deriva de la medida F [Manning and Schütze 01]. De este modo tenemos que:

$$recall = \frac{|\, A \cap S \,|}{|\, S \,|}, \quad precision = \frac{|\, A \cap P \,|}{|\, A \,|},$$
$$AER(S, P; A) = 1 - \frac{|\, A \cap S \,| + |\, A \cap P \,|}{|\, A \,| + |\, S \,|}$$

Por lo tanto, un error de *recall* solamente podrá ocurrir si no se encuentra un alineamiento $S(eguro)$, y un error de *precision* podrá ocurrir si el alineamiento encontrado ni tan siquiera es $P(osible)$.

El conjunto de pares de frases para los cuales se llevó a cabo el alineamiento manual fue seleccionado aleatoriamente del corpus de entrenamiento. Cabe destacar que el entrenamiento se realizó de forma completamente no supervisada, es decir no se utilizaron alineamientos manuales durante el entrenamiento. Desde este punto de vista por lo tanto no es necesario tener un corpus de test separado.

4.7.2 Resultados

De acuerdo a la experimentación preliminar llevada a cabo con respecto a la calidad de los alineamientos de la versión integrada de los modelos por ME con

[3] Del inglés Alignment Error Rate.

respecto a la simplificación propuesta en la sección anterior, podemos concluir que la diferencia de calidad obtenida es despreciable. Por lo tanto, y dado el gran coste computacional que la versión integrada conlleva, los resultados aquí presentados se realizaron utilizando la versión simplificada.

Las tablas 4.8 y 4.9 muestran la calidad de los alineamientos (en términos de *AER*) para diferentes tamaños de corpus de entrenamiento para las tareas HANSARDS y VERBMOBIL respectivamente[4]. En ambas tablas podemos ver el valor del *AER* de forma comparativa entre un entrenamiento convencional y un entrenamiento utilizando la integración de modelos por máxima entropía. Como podemos ver la experimentación se llevo a cabo para diferentes esquemas de entrenamiento. El esquema de entrenamiento se define de acuerdo al número de iteraciones realizadas por cada modelo. A título de ejemplo el esquema $1^5 2^5 3^3 4^3$ indica que se realizaron 5 iteraciones de modelo 1, 5 del modelo 2, 3 del modelo 3 y 3 del modelo 4. La elección de los esquemas de entrenamiento se realizó de forma empírica, estableciendo ciertos parámetros de entrenamiento. En ambas tablas podemos ver los mejores resultados obtenidos.

Los resultados a nivel de *precision* y *recall* con y sin tener en cuenta el entrenamiento por ME se muestran en las figuras 4.2, 4.3 y 4.4, 4.5 para las tareas HANSARDS y VERBMOBIL respectivamente. En la leyenda de todas estas figuras, `Base` se refiere a un entrenamiento convencional, y `MaxEnt` se refire al entrenamiento integrando los modelos por máxima entropía.

A la vista de figuras de *precision* y *recall* se puede observar que, en todos los casos, se obtiene mayor *precision* y mayor *recall* cuando se usan los modelos contextuales por ME. También se puede observar que las diferencias son mínimas para tamaños de corpus de entrenamiento pequeños, siendo prácticamente idénticos para el caso de 500 frases de entrenamiento.

A la vista de los resultados podemos ver que el ratio de errores de alineamiento mejora cuando utilizamos los modelos léxicos dependientes del contexto. Para la tarea VERBMOBIL la mejora es menor que para la tarea HANSARDS, lo cual se debe al hecho de que el resultado base de calidad en el alineamiento es bastante bueno. Se puede ver que se obtienen mejores alineamientos para los modelos más simples, de hecho cuanto mejor es el modelo más difícil resulta mejorar los resultados.

Como era de esperar, el entrenamiento por ME tiene mayor importancia cuanto mayor es el tamaño del corpus a utilizar. Para el corpus más pequeño (500 frases de entrenamiento), el número de eventos de entrenamiento para los modelos por ME es muy pequeño, por lo tanto no es posible desambiguar algunas traducciones/alineamientos para diferentes contextos. A mayor tamaño del corpus mejores resultados, por lo tanto, esperamos obtener incluso mejores resultados cuando se usen corpus más grandes.

Por otra parte y para dar una idea de cómo de buenos pueden ser los

[4] El hecho de utilizar estas dos tareas para esta experimentación es debido a la disponibilidad del esquema de anotación para ellas y no para otras tareas.

Esquema de entrenamiento	Modelo	Tamaño del corpus		
		0.5K	8K	128K
1^5	1	48.0	35.1	29.2
	1+ME	47.7	32.7	22.5
$1^5 2^5$	2	46.0	29.2	21.9
	2+ME	44.7	28.0	19.0
$1^5 2^5 3^3$	3	43.2	27.3	20.8
	3+ME	42.5	26.4	17.2
$1^5 2^5 3^3 4^3$	4	41.8	24.9	17.4
	4+ME	41.3	24.3	14.1
$1^5 2^5 3^3 4^3 5^3$	5	41.5	24.8	16.2
	5+ME	41.2	24.3	14.3

Tabla 4.8: AER [%] para la tarea HANSARDS para distintos tamaños de corpus de entrenamiento.

Esquema de entrenamiento	Modelo	Tamaño del corpus		
		0.5K	8K	34K
1^5	1	27.7	19.2	17.6
	1+ME	24.6	16.6	13.7
$1^5 2^5$	2	26.8	15.7	13.5
	2+ME	25.3	14.1	10.8
$1^5 2^5 3^3$	3	25.6	13.7	10.8
	3+ME	24.1	11.6	8.8
$1^5 2^5 3^3 4^3$	4	23.6	10.0	7.7
	4+ME	22.8	9.3	7.0
$1^5 2^5 3^3 4^3 5^3$	5	22.6	9.9	7.2
	5+ME	22.3	9.6	6.8

Tabla 4.9: AER [%] para la tarea VERBMOBIL para distintos tamaños de corpus de entrenamiento.

parámetros obtenidos con la integración de los modelos por ME en el entrenamiento, en las tablas 4.10 y 4.11 se muestra la perplejidad del conjunto de entrenamiento para los entrenamientos realizados, para las tareas HANSARDS y VERBMOBIL respectivamente. Estos resultados están en consonancia con los resultados de perplejidad obtenidos en la sección anterior, de manera que de alguna manera corroboran los resultados anteriores.

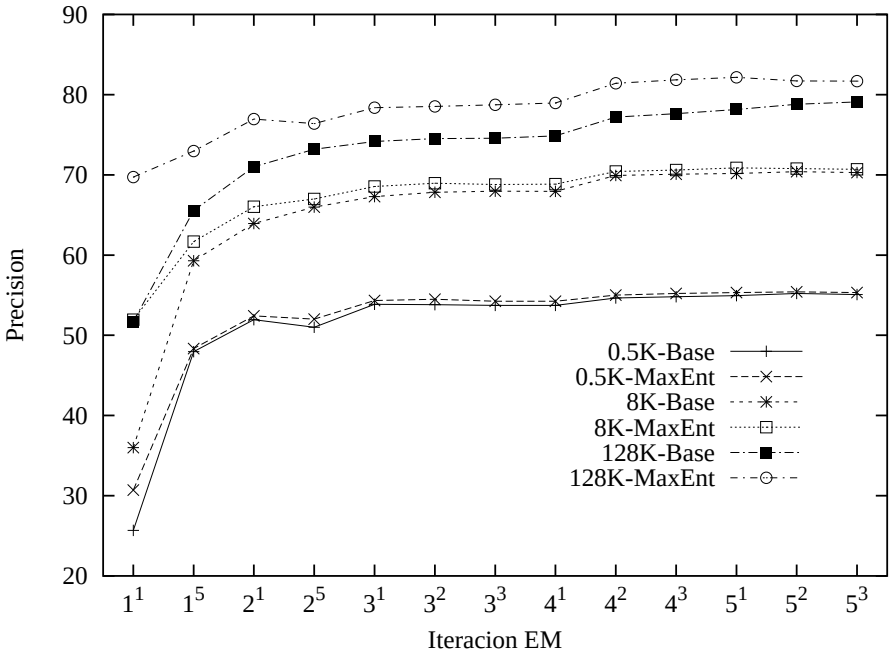


Figura 4.2: *Precision* en [%] para la tarea HANSARDS para diferentes tamaños de corpus de entrenamiento, para cada iteración del algoritmo EM.

4.8 Conclusiones

En este capítulo hemos definido modelos léxicos dependientes del contexto dentro marco de la máxima entropía. Para ello hemos definido las características relevantes al problema a modelar, así como cómo realizar un entrenamiento de los modelos por máxima entropía a partir de un corpus bilingüe alineado a nivel de palabra. Del mismo modo hemos llevado a cabo un estudio experimental para hacernos una idea de como puede influir el uso de modelos ME en sistemas de traducción existentes.

Además hemos visto cómo integrar los modelos dependientes del contexto basados en máxima entropía en el entrenamiento por máxima verosimilitud de modelos estadísticos de alineamiento. Hemos estudiado experimentalmente dicha integración, llevando a cabo una evaluación de la calidad de los alineamientos obtenidos con el nuevo esquema de entrenamiento comprándolos con los resultados base obtenidos mediante modelos convencionales. Como podemos ver en la sección 4.7.2, en todos los casos hemos obtenido mejor calidad en los alineamientos usando los modelos léxicos dependientes del contexto. Estos resultados son directamente comparables con los presentados en [Och and Ney 03], siendo estos prácticamente idénticos a los aquí presentados en los casos en que se siguen

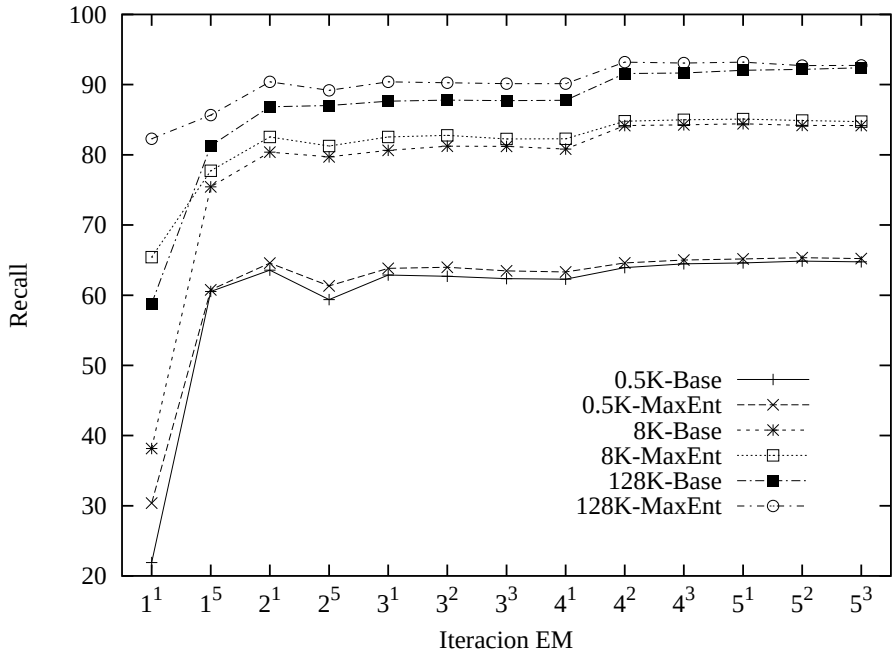


Figura 4.3: *Recall* en [%] para la tarea HANSARDS para diferentes tamaños de corpus de entrenamiento, para cada iteración del algoritmo EM.

mismos esquemas de entrenamiento, sin utilizar los modelos basados en máxima entropía. Esto nos hace pensar que los resultados aquí presentados son correctos.

Un hecho a destacar en el trabajo mencionado es que el uso del modelo HMM, en lugar del modelo 2 en el esquema de entrenamiento mejora sustancialmente los resultados. Esto nos inclina a que en el futuro realicemos también la integración de los modelos por ME en el entrenamiento de los modelos HMM, con lo que presumiblemente se obtengan incluso mejores resultados que los aquí expuestos.

Haciendo un estudio minucioso de los errores más comunes de alineamiento, nuestro plan para el futuro pasa por incluir características más discriminantes que proporcionen grandes mejoras, como por ejemplo dependencias con otras palabras fuente y destino, etiquetas POS y constituyentes sintácticos. También entra dentro de nuestros planes diseñar modelos por ME para los modelos de fertilidad y de distorsión, los cuales permitirán integrar fácilmente mayor número de dependencias, como modelos de distorsión/alineamiento de segundo orden sin caer en problemas de estimación de gran cantidad de parámetros como ocurre con los modelos convencionales. También esperamos obtener mejoras realizando un modelado específico para las palabras raras (o infrecuentes), las cuales no hemos tenido en cuenta en los modelos por ME aquí presentados.

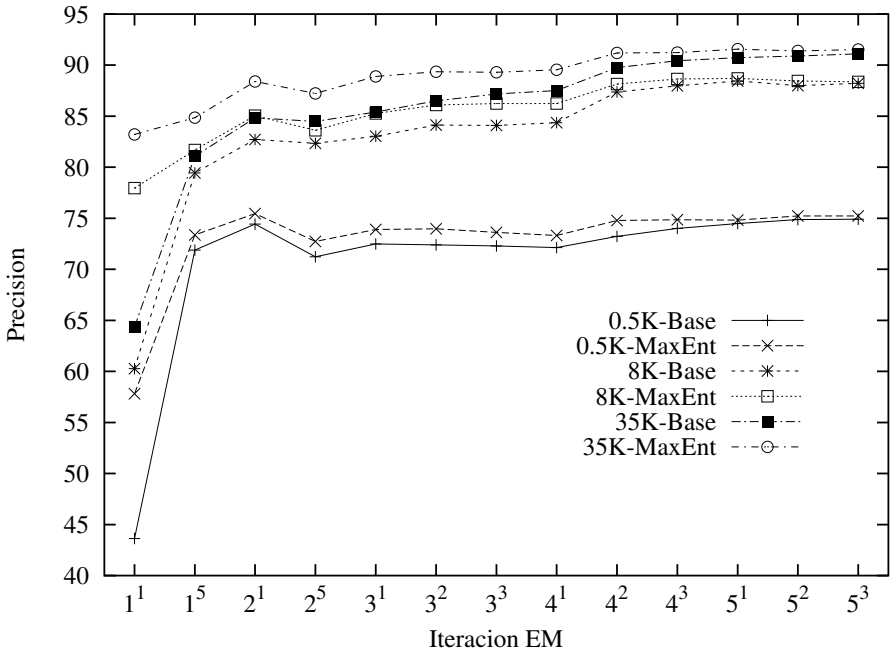


Figura 4.4: *Precision* en [%] para la tarea VERBMOBIL para diferentes tamaños de corpus de entrenamiento, para cada iteración del algoritmo EM.

Esquema de entrenamiento	Modelo	Tamaño del corpus		
		0.5K	8K	128K
1^5	1	34.7	47.3	72.9
	1+ME	33.9	43.4	51.9
$1^5 2^5$	2	11.1	25.5	40.2
	2+ME	10.5	23.5	30.9
$1^5 2^5 3^3$	3	21.3	43.3	69.8
	3+ME	20.8	39.2	49.8
$1^5 2^5 3^3 4^3$	4	47.4	72.8	104.1
	4+ME	46.1	65.7	68.1
$1^5 2^5 3^3 4^3 5^3$	5	8.2	13.2	19.9
	5+ME	8.1	11.9	14.1

Tabla 4.10: Perplejidad del conjunto de entrenamiento para la tarea HANSARDS, para distintos tamaños de corpus de entrenamiento.

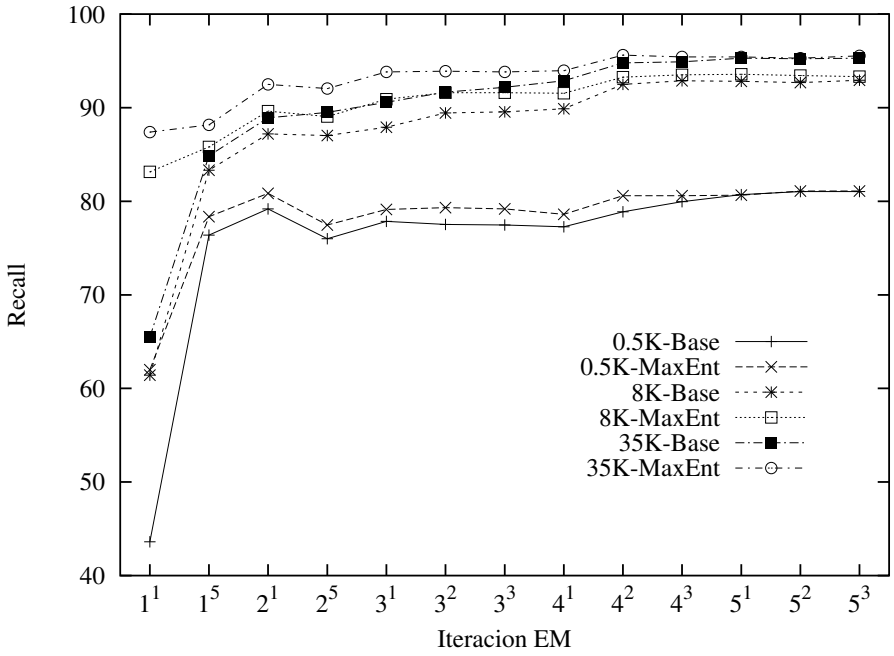


Figura 4.5: *Recall* en [%] para la tarea VERBMOBIL para diferentes tamaños de corpus de entrenamiento, para cada iteración del algoritmo EM.

Esquema de entrenamiento	Modelo	Tamaño del corpus		
		0.5K	8K	34K
1^5	1	22.2	27.6	29.9
	1+ME	20.7	24.0	24.5
$1^5 2^5$	2	7.7	14.6	16.9
	2+ME	7.1	13.0	14.2
$1^5 2^5 3^3$	3	12.9	23.1	26.8
	3+ME	11.7	19.8	20.2
$1^5 2^5 3^3 4^3$	4	24.0	32.9	35.0
	4+ME	22.0	27.7	26.9
$1^5 2^5 3^3 4^3 5^3$	5	5.3	7.1	7.7
	5+ME	4.9	6.1	6.1

Tabla 4.11: Perplejidad del conjunto de entrenamiento para la tarea VERBMOBIL para distintos tamaños de corpus de entrenamiento.

III

Algoritmos de búsqueda

Capítulo 5

Algoritmos de búsqueda basados en programación dinámica

COMO hemos comentado en el capítulo de introducción, el objetivo de la traducción automática estadística consiste en resolver la maximización dada en la fórmula (1.2). En esta parte de la tesis veremos distintas formas de resolver esa maximización utilizando diferentes técnicas. En este capítulo expondremos una serie de algoritmos de búsqueda basados en la técnica de programación dinámica para la resolución de problemas.

5.1 Introducción

En primer lugar expondremos un algoritmo de búsqueda heurístico diseñado específicamente para el modelo 2. A continuación veremos una versión iterativa del algoritmo basado en un proceso iterativo de refinamiento de la solución, en el cual podemos obtener una mejor solución a partir de la solución obtenida en una iteración previa. Por último veremos una posible ampliación a estos algoritmos para poder trabajar con los modelos más complejos de IBM, concretamente para los modelos 3, 4 y 5, y con el modelo HMM.

5.2 Algoritmo básico de búsqueda para el Modelo 2

Como ya hemos comentado, el problema de la búsqueda consiste en diseñar un algoritmo eficiente para buscar $\hat{e}_1^{\hat{I}}$ (o una aproximación de $\hat{e}_1^{\hat{I}}$).

La ecuación (1.2) podemos reescribirla como:

$$\begin{aligned} \hat{e}_1^{\hat{I}} &= \arg\max_{I,e_1^I}\{Pr(e_1^I)\cdot Pr(f_1^J \mid e_1^I)\} \\ &= \max_I \arg\max_{e_1^I}\{Pr(e_1^I)\cdot Pr(f_1^J \mid e_1^I)\} \end{aligned} \tag{5.1}$$

En otras palabras, la maximización en (1.2) se puede realizar buscando la mejor cadena de salida e_1^I para cada posible longitud I, y buscando ese I óptimo.

Vamos a suponer que la longitud de la cadena de salida es conocida (llamémosle simplemente I). El modelo de traducción será el modelo 2 y el modelo de lenguaje (por comodidad en la notación) un modelo de bigramas ($p(\cdot)$). De este modo obtenemos el siguiente criterio de búsqueda[1]:

$$\begin{aligned} \hat{e}_1^I &= \arg\max_{e_1^I}\{Pr(e_1^I)\cdot Pr(f_1^J \mid e_1^I)\} \\ &= \arg\max_{e_1^I}\left\{\prod_{i=1}^{I} p(e_i \mid e_{i-1})\cdot \prod_{j=1}^{J}\sum_{i=0}^{I} t(f_j \mid e_i)\cdot a(i \mid j,J,I)\right\} \end{aligned} \tag{5.2}$$

Esta maximización se puede resolver eficientemente, aunque de forma aproximada, mediante programación dinámica definiendo las siguientes cantidades auxiliares:

- $Q(e,i,j)$: Esta cantidad acumula para cada posible posición j (o palabra) de la frase de entrada, la contribución del modelo de traducción para toda posible palabra destino e que pueda aparecer en la posición i, para una determinada hipótesis parcial e_1^{i-1}. Se define como:

$$Q(e,i,j) = t(f_j \mid e_i)\cdot a(i \mid j,J,I) + \sum_{k=0}^{i-1} t(f_j \mid e_k)\cdot a(k \mid j,J,I) \tag{5.3}$$

- $T(e,i)$: Esta cantidad acumula la contribución del modelo de lenguaje de una hipótesis parcial e_1^{i-1} cuando la palabra e ocupa esa posición i. Se define como:

$$T(e,i) = p(e \mid e_{i-1}) \prod_{k=1}^{i-1} t(e_k \mid e_{k-1}) \tag{5.4}$$

[1] Dado que consideramos conocida la longitud de la cadena de salida I no tiene sentido incluir en el criterio de búsqueda el término $\epsilon(I|J)$ que aparece en la fórmula (3.9)

Por tanto la maximización (5.2) la podemos reescribir como:

$$\hat{e}_1^I = \arg\max_{e_1^I} \left\{ T(e_I \mid I) \cdot \prod_{j=1}^{J} Q(e_I, I, j) \right\} \tag{5.5}$$

Y para resolverla mediante programación dinámica establecemos la siguiente recursión:

$$\begin{aligned} Q(e,i,j) &= Q(\hat{e}(e,i), i-1, j) + t(f_j \mid \hat{e}(e,i)) \cdot a(i \mid j, J, I) && (5.6) \\ T(e,i) &= T(\hat{e}(e,i), i-1) \cdot p(e \mid \hat{e}(e,i)) && (5.7) \end{aligned}$$

donde la expresión $\hat{e}(e,i)$ se define como:

$$\hat{e}(e,i) = \arg\max_{e' \in \mathcal{E}} \left\{ p(e \mid e') \cdot T(e', i-1) \cdot \prod_{j=1}^{J} \left(Q(e', i-1, j) + t(f_j \mid e) \cdot a(i \mid j, J, I) \right) \right\} \tag{5.8}$$

La base de la recursión $\forall e \in \mathcal{E}$ y $\forall j : 1 \leq j \leq J$ será:

$$\begin{aligned} Q(e,1,j) &= t(f_j \mid e_0) \cdot a(0 \mid j, J, I) && (5.9) \\ T(e,1) &= 1.0 && (5.10) \end{aligned}$$

En el algoritmo 5.1 podemos ver como quedaría este proceso de búsqueda de forma algorítmica.

Todo lo comentado para el modelo 2 es también aplicable para el modelo 1, pues, como ya se ha comentado antes, éste no es más un caso particular del primero en el que la probabilidad de alineamiento $a(\cdot)$ pasa a ser una constante. Dado que en toda la formulación anterior estamos maximizando, para adaptar el algoritmo *DPSearchM2* al modelo 1 bastaría con eliminar de la formulación anterior los términos $a(i \mid j, J, I)$.

5.2.1 Detalles de implementación

Dada la relativa complejidad del algoritmo cabe realizar ciertas matizaciones acerca de la implementación que hemos llevado a cabo.

La implementación de este algoritmo consiste en la construcción de una malla o *trellis* de búsqueda de izquierda a derecha, que desde el punto de vista de la programación dinámica consiste en un grafo multietapa formado por tantas etapas como posiciones haya que generar en la cadena de salida. En la etapa i del proceso de búsqueda tendremos todas las posibles hipótesis parciales de longitud i, de modo que el proceso de búsqueda acabará cuando hayamos alcanzado la etapa I. En ese instante la solución al problema será aquella hipótesis

Algoritmo: ***DPSearchM2***
entrada: $I, t(\cdot), a(\cdot), f_1^J$
salida: $\hat{e}_1^I$
  //Inicialización
  para_todo $e \in \mathcal{E}$ hacer
    para $j = 1 \cdots J$ hacer
      $Q(e, 1, j) = t(f_j \mid e_0) \cdot a(0 \mid j, J, I)$
    fin_para
    $T(e, 1) = 1.0$
  fin_para
  //Método
  para $i = 1 \cdots I$ hacer
    para_todo $e \in \mathcal{E}$ hacer
      Calcular $\hat{e}(e, i)$ mediante la ecuación (5.8)
      para $j = 1 \cdots J$ hacer
        Actualizar $Q(e, i, j)$ de acuerdo a (5.6)
      fin_para
      Actualizar $T(e, i)$ de acuerdo a (5.7)
    fin_para
  fin_para
  devolver: $H(\hat{e}_I)$, t.q.: $\hat{e}_I = \arg\max_{e_I \in \mathcal{E}} \left\{ T(e_I, I) \prod_{j=1}^{J} Q(e_I, I, j) \right\}$

Algoritmo 5.1: Algoritmo básico de búsqueda para el modelo 2.

que tenga asignada la mejor *puntuación* de acuerdo al criterio de optimización elegido. Cada estado del *trellis* quedará definido por el par (e, i), el cual definirá unívocamente cada hipótesis parcial (o total en el caso $i = I$), de modo que:

$$e_I = \arg\max_{e_I \in \mathcal{E}} \left\{ T(e_I, I) \prod_{j=1}^{J} Q(e_I, I, j) \right\}$$

define la mejor solución al problema, la asociada al estado (e_I, I). Teniendo en cuenta esto podemos definir una función $H(e_I)$ de modo que devuelva la hipótesis asociada a ese estado.

A continuación, y de cara a entender el resto de algoritmos que se proponen en sucesivas secciones, veremos mediante un ejemplo gráfico como funciona el algoritmo en un momento dado de la búsqueda, es decir en el paso i del proceso de construcción del trellis:

- En la figura 5.1 podemos ver como sería una hipotética situación inicial en el momento de realizar una decisión en el proceso de búsqueda. En este punto se trata de encontrar la mejor hipótesis parcial a almacenar en el estado (e, i) del trellis. Por comodidad, supongamos que en el paso $i - 1$

sólo existen tres hipótesis activas, las que forman los estados $(e', i-1)$, $(e'', i-1)$ y $(e''', i-1)$. Cada estado tendrá sus correspondientes valores de $Q(e^*, i-1, j), \forall j : 1 \leq j \leq J$ y de $T(e^*, i-1)$.

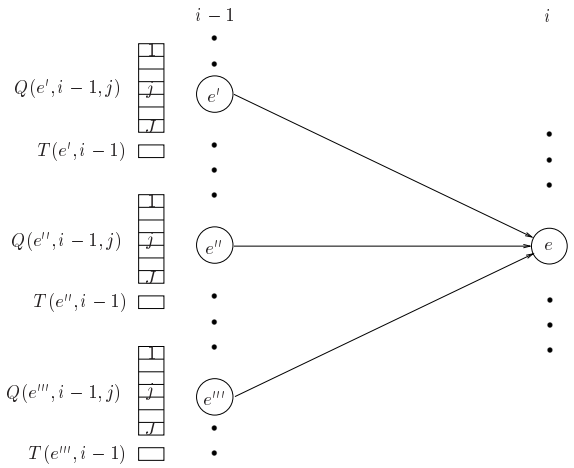


Figura 5.1: Situación inicial en el momento de decidir la mejor hipótesis parcial para el estado (e, i) del trellis de búsqueda.

- En la figura 5.2 vemos como afecta, a la contribución de la *puntuación* de la hipótesis, la utilización de las diferentes hipótesis parciales previas. El criterio de decisión viene dado por el valor de la expresión:

$$prod(e^*) = T(e^*, i-1) \cdot p_b(e \mid e^*) \times \prod_{j=1}^{J} \left(Q(e^*, i-1, j) + t(f_j \mid e) \cdot a(i \mid j, J, I) + R(j, i+1) \right) ,$$

  En esa figura también podemos ver cómo se calcula ese valor para el caso concreto de e''.

- En la figura 5.3 se toma la decisión de transitar del estado $(e'', i-1)$ al estado (e, i), por lo tanto habrá que actualizar convenientemente los valores de $Q(e, i, j)$ $\forall j : 1 \leq j \leq J$, y de $T(e, i)$ como se muestra en la figura. Una vez alcanzados los estados (e_I^*, I) se elegirá aquel que proporcione el mayor valor de $prod(e_I^*)$ y por tanto esa será la mejor hipótesis $(\tilde{e}_1^I)$ para la longitud I.

En todo momento hemos considerado que la longitud óptima de la cadena de salida era I. Evidentemente esto es mucho suponer en la práctica, de modo

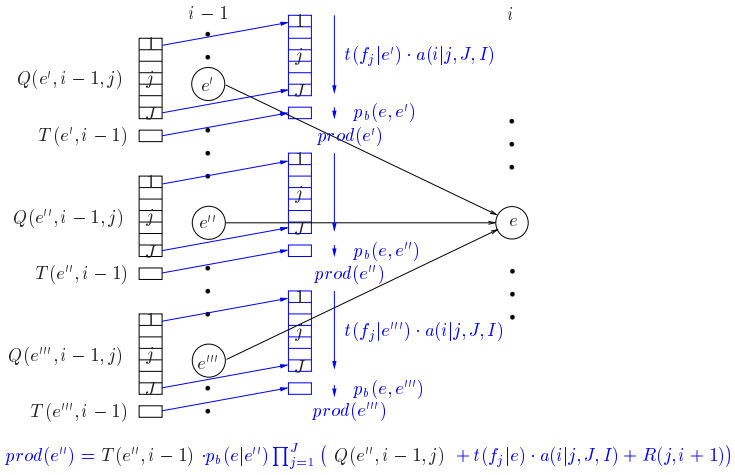


Figura 5.2: Cálculo del criterio de búsqueda para el estado (e, i) de acuerdo a los posibles estados previos en el trellis (en este caso solo se consideran 3 posibles hipótesis previas por claridad).

que en la implementación de este algoritmo deberemos solucionar el problema planteado en la ecuación (5.1), es decir, además de buscar cual es la solución óptima (o una aproximación a ella) para una longitud dada, deberemos buscar que longitud maximiza la probabilidad de la ecuación (5.1). La solución a esto pasa por realizar una serie de ejecuciones del algoritmo para un subconjunto de posibles longitudes $I = 1 \cdots I_{max}$. Para calcular ese subconjunto se utiliza la distribución de probabilidad de longitudes definida para el modelo 2 $\epsilon(I \mid J)$ estimada previamente en el entrenamiento. En este caso ya estaremos en disposición también de añadir el término correspondiente a $\epsilon(I \mid J)$ al criterio de optimización (5.2), que previamente habíamos obviado por considerar conocida la longitud de la cadena de salida.

Más consideraciones de implementación acerca de reducciones del espacio de búsqueda y eficiencia computacional serán consideradas en la sección 5.7.

5.2.2 Aproximación por maximización

Normalmente, el criterio de búsqueda de la ecuación (5.2) anterior se relaja, de modo que se intenta evitar tener que realizar una suma para toda posición de salida i, con lo que en la práctica se sigue la aproximación por maximización. En este caso realizar una maximización en vez de una suma pasa por establecer un criterio de búsqueda distinto. De este modo el criterio de búsqueda queda:

$$\hat{e}_1^I = \arg\max_{e_1^I} \left\{ \prod_{i=1}^{I} p(e_i \mid e_{i-1}) \cdot \max_{i:0 \le i \le I} \prod_{j=1}^{J} t(f_j \mid e_i) \cdot a(i \mid j, J, I) \right\} \qquad (5.11)$$

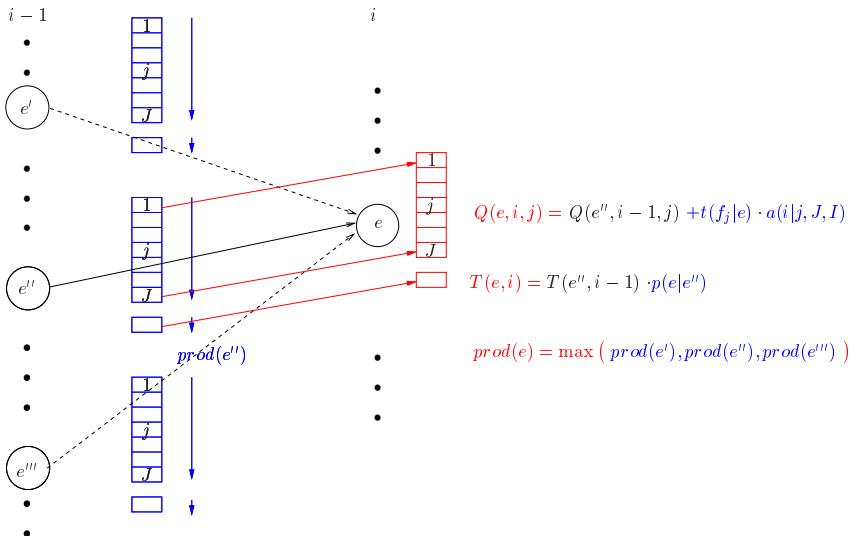


Figura 5.3: Actualización de los valores de $Q(e,i,j)$ y $T(e,i)$ para el estado (e,i) del trellis, una vez realizada la decisión.

Por tanto habrá que redefinir convenientemente todas las expresiones en las que intervenga el modelo de traducción, concretamente:

- La expresión (5.3) pasará a ser:

$$Q(e,i,j) = \max\Big\{t(f_j \mid e_i)\cdot a(i \mid j,J,I), \max_{k=0\cdots i-1}\big(t(f_j \mid e_k)\cdot a(k \mid j,J,I)\big)\Big\} \tag{5.12}$$

- Y para el caso de la expresión (5.8) tendríamos:

$$\hat{e}(e,i) = \arg\max_{e'\in\mathcal{E}}\left\{p(e \mid e')\cdot T(e',i-1)\cdot \prod_{j=1}^{J}\max\big(Q(e',i-1,j), t(f_j \mid e)\cdot a(i \mid j,J,I)\big)\right\} \tag{5.13}$$

5.3 Versión iterativa del algoritmo *DPSearchM2*

Según hemos planteado el algoritmo de la sección anterior no podemos garantizar la optimalidad de la solución según el criterio de búsqueda propuesto

en la expresión (5.2). Esto es debido a que la maximización hay que hacerla teniendo en cuenta toda posición i de la hipótesis de salida. Esto es un problema grave dado que la búsqueda se basa en la construcción de hipótesis parciales (de izquierda a derecha) para cada posición de la cadena de salida. Esto deriva en que el algoritmo planteado sea una aproximación heurística a la solución dado que, en cada paso del trellis de búsqueda, se subestima el coste de la aportación que el modelo de traducción hace a la *puntuación* final de la hipótesis.

La obtención de la solución óptima a la expresión (5.2) pasaría por poder calcular de forma exacta la cantidad:

$$Q(e,i,j) = t(f_j \mid e_i) \cdot a(i \mid j,J,I) + \sum_{k=0;/k\neq i}^{I} t(f_j \mid e_k) \cdot a(k \mid j,J,I) \ , \qquad (5.14)$$

aunque, como ya hemos comentado, cabe recordar que el problema de la búsqueda para modelos que permitan reordenamientos es NP-completo [Knight 99a].

Según está planteado el algoritmo este problema tiene difícil solución dado que el sufijo e_{i+1}^{I} se desconoce durante el proceso de búsqueda. En lo que sigue desarrollaremos otro algoritmo de búsqueda, también heurístico (aunque como veremos más adelante se comporta bien en la práctica) y basado en el anterior, en el que se realizará una estimación del coste de completar una hipótesis. Esta estimación del coste de completar una hipótesis se realizará mediante un proceso de refinamiento iterativo, en el que en cada iteración se considerará la hipótesis obtenida en la iteración previa como sufijo de la hipótesis parcial que se esté construyendo.

Todo esto conlleva a reescribir la expresión (5.8), que define el procedimiento recursivo del algoritmo *DPSearchM2*, del siguiente modo:

$$\hat{e}(e,i) = \arg\max_{e' \in \mathcal{E}} \left\{ p(e \mid e') \cdot T(e', i-1) \cdot \prod_{j=1}^{J} \left(Q(e', i-1, j) + t(f_j \mid e) \cdot a(i \mid j,J,I) + R(j, i+1) \right) \right\} \qquad (5.15)$$

donde $R(j,i)$ se define como:

$$R(j,i) = \sum_{k=i}^{I} t(f_j \mid \tilde{e}_k) a(k \mid j,J,I) \qquad (5.16)$$

y $\tilde{e}_1^I$ es una aproximación a la solución óptima (obtenida en una iteración anterior).

Con todo esto el algoritmo *IterativeDPSearchM2* queda como se muestra en el algoritmo 5.2. Evidentemente en la primera iteración de este algoritmo

no tendremos información de una hipótesis previa, de modo que en ese caso $R(j,i) = 0, \forall i : 1 \leq i \leq I$ y $\forall j : 1 \leq j \leq J$, es decir, la primera hipótesis del proceso iterativo será la hipótesis que se obtiene con el algoritmo básico *DPSearchM2*.

En este sentido existe la posibilidad de ayudar al algoritmo de búsqueda desde la primera iteración. Esto es estableciendo un valor a $R(j,i)$ distinto de cero de forma heurística pero a un valor que nunca subestime su valor real. Así podríamos inicializar $R(j,i)$, $\forall i : 1 \leq i \leq I$ y $\forall j : 1 \leq j \leq J$ del siguiente modo:

$$R(i,j) = \sum_{k=i}^{I} \max_{e} \{t(e \mid f_j) \cdot a(i \mid j, J)\}$$

Cuando hagamos uso de esta inicialización heurística en la sección de experimentación diremos que estaremos usando *inicialización heurística*.

De forma análoga se podría definir el algoritmo *IterativeDPSearchM1*, teniendo en cuenta las consideraciones de la sección anterior.

5.4 Aproximación por Viterbi para los modelos 3, 4 y 5

En las dos secciones previas hemos presentado dos algoritmos basados en programación dinámica que resuelven de forma heurística el problema de la búsqueda para los modelos 1 y 2. En esta sección veremos de qué forma podemos introducir en el algoritmo *IterativeDPSearchM2* la información (en principio más completa) que nos proporcionan los modelos más complejos de IBM sin necesidad de modificar la filosofía del algoritmo. El hecho radica precisamente en la naturaleza iterativa del algoritmo.

Según hemos comentado anteriormente el algoritmo *IterativeDPSearchM2* se basa en un procedimiento de refinamiento iterativo en el que la información de la hipótesis óptima (de acuerdo a un criterio de búsqueda) de una iteración previa del algoritmo puede utilizarse como fuente de información para refinar la solución en la iteración siguiente.

La inclusión de los modelos más complejos de IBM se realiza precisamente en el momento de finalizar la búsqueda y consiste en que una vez alcanzados los posibles estados finales $\{(e_I, I) : e_I \in \mathcal{E}\}$ (que en principio habrá tantos como la talla del vocabulario de salida $|\mathcal{E}\ |$), en vez de elegir como hipótesis aquella que determina el criterio de búsqueda, elegir la hipótesis que mejor alineamiento por Viterbi produzca para un modelo en concreto. Según lo comentado hasta ahora no es posible establecer este criterio de búsqueda durante el proceso de decisión dado que para poder calcular el alineamiento por Viterbi entre un par de frases es necesario conocer ambas frases por completo. Es precisamente este motivo el porqué de retrasar esa decisión hasta el momento del proceso de búsqueda en el que las posibles hipótesis contienen todas sus palabras.

Algoritmo: ***IterativeDPSearchM2***
entrada: $I, t(\cdot), a(\cdot), f_1^J$
salida: $\hat{e}_1^I$
  //*Inicialización*
  Sea $\tilde{e}_1^J$ la solución al algoritmo ***DPSearchM2***
  //*Iteración*
  mientras (convergencia) hacer
    ${e'}_1^I = \tilde{e}_1^J$ {Actualización de hipótesis previa}
    para $i = 1 \cdots I$ hacer
      para $j = 1 \cdots J$ hacer
        $R(j,i) = t(f_j \mid e'_i) \cdot a(i \mid j, J, I)$ {Actualización de la matriz R}
      fin_para
    fin_para
    //*Método*
    para $i = 1 \cdots I$ hacer
      para_todo $e \in \mathcal{E}$ hacer
        Calcular $\hat{e}(e,i)$ mediante la ecuación (5.15)
        para $j = 1 \cdots J$ hacer
          Actualizar $Q(e,i,j)$ de acuerdo a (5.6)
        fin_para
        Actualizar $T(e,i)$ de acuerdo a (5.7)
      fin_para
    fin_para
    $\tilde{e}_1^I = H(\hat{e}_I)$, t.q.: $\hat{e}_I = \arg\max_{e_I \in \mathcal{E}} \left\{ T(e_I, I) \prod_{j=1}^{J} Q(e_I, I, j) \right\}$
  fin_mientras
  devolver: $\tilde{e}_1^I$

Algoritmo 5.2: Algoritmo iterativo de búsqueda para el modelo 2.

Evidentemente esta solución es una solución aproximada pues el proceso de decisión (o construcción del trellis de búsqueda) se seguirá haciendo de acuerdo al criterio de búsqueda de la expresión (5.2) para el modelo 2. De cualquier forma, y como veremos en la sección de experimentación de este tema, esta aproximación ayuda a obtener mejores resultados de traducción y ello radica precisamente en que al considerar la información de los modelos más complejos de IBM en la selección de las mejores hipótesis en cada iteración, usamos más información y por tanto el algoritmo es capaz de discernir mejor entre hipótesis. Esa mejor elección de hipótesis temporales por iteración ayuda a mejorar el funcionamiento del algoritmo en iteraciones posteriores, pues la estimación del resto del coste de completitud de hipótesis parciales mejora.

En definitiva, para introducir esta aproximación dentro del algoritmo *IterativeDPSearchM2* habrá que seguir los siguientes pasos:

1. Realizar una primera iteración utilizando el algoritmo *DPSearchM2* hasta el paso I del proceso de búsqueda.

2. Para cada hipótesis almacenada en el conjunto de estados $\{(e_I, I) : e_I \in \mathcal{E}\}$ (es decir, $\forall e_1^I \in \{H(e_I) : e_I \in \mathcal{E}\}$): calcular el alineamiento por Viterbi (de acuerdo al modelo $m \in \{3, 4, 5\}$) con la frase de entrada. Esto es:

$$\tilde{e}_1^I = \underset{e_1^I \in \{H(e_I) : e_I \in \mathcal{E}\}}{\arg\max} \left\{ V(f_1^J \mid e_1^I; m) \right\}$$

3. Actualizar el array R de acuerdo a $\tilde{e}_1^I$, y volver al paso 1 para realizar otra iteración del algoritmo *IterativeDPSearchM2*.

Esto mismo, de forma algorítmica, lo podemos ver en el algoritmo 5.3, al cual hemos llamado *ViterbiIterativeDPSearchM(m)*, donde m indicará el modelo a utilizar para calcular el alineamiento por Viterbi.

Cabe matizar que según se comentó en el capítulo 3 no existe forma práctica para calcular el alineamiento por Viterbi para los modelos 3, 4 y 5. De hecho el alineamiento para estos modelos se realiza utilizando un método de ascensión de colina[2], el cual partiendo del alineamiento por Viterbi según el modelo 2 lo modifica durante una serie de iteraciones, realizando pequeñas modificaciones al alineamiento (normalmente cambios y movimientos), siempre y cuando la probabilidad del alineamiento ($Pr(\mathbf{a} \mid \mathbf{e}, \mathbf{f})$) mejore. Es precisamente este hecho el que nos inspiró en el diseño de este algoritmo, dado que el proceso de decisión/búsqueda se realiza de acuerdo al criterio de búsqueda establecido por el modelo 2, al igual que para calcular el alineamiento, y del mismo modo la decisión final de la mejor hipótesis se realiza de acuerdo a otros modelos más complejos.

Aunque en esta sección nos hemos limitado a hablar de los modelos 3, 4 y 5, este algoritmo se puede hacer extensivo al resto de modelos, es decir, sería perfectamente posible obtener el alineamiento por Viterbi para los modelos 1, 2 y HMM. De todas formas, es de esperar que se obtengan mejores resultados con los modelos más complejos. En la sección 5.8.2 llevaremos a cabo un experimento para comparar la influencia de cada uno de los modelos en el algoritmo *ViterbiIterativeDPSearchM(m)*.

5.5 Más sobre algoritmos de búsqueda basados en programación dinámica

Existen otros algoritmos de búsqueda que siguen la técnica de programación dinámica. Concretamente, en [Tillmann 01] podemos encontrar algoritmos para el modelo HMM (ver sección 3.4.1) y para versiones ligeramente modificadas de

[2] En inglés *hillclimbing*

Algoritmo: ***ViterbiIterativeDPSearchM(m)***
entrada: $I, t(\cdot), a(\cdot), f_1^J$
salida: $\hat{e}_1^I$
  //*Inicialización*
  Ejecutar el algoritmo ***DPSearchM2*** hasta el paso I de la búsqueda.
  Sea $\tilde{e}_1^I = \arg\max_{e_1^I \in \{H(e_I): e_I \in \mathcal{E}\}} \{V(f_1^J \mid e_1^I; m)\}$
  //*Iteración*
  mientras (convergencia) hacer
    ${e'}_1^I = \tilde{e}_1^J$ {Actualización de hipótesis previa}
    para $i = 1 \cdots I$ hacer
      para $j = 1 \cdots J$ hacer
        $R(j,i) = t(f_j \mid e'_i) \cdot a(i \mid j, J, I)$ {Actualización de la matriz R}
      fin_para
    fin_para
    //*Método*
    para $i = 1 \cdots I$ hacer
      para_todo $e \in \mathcal{E}$ hacer
        Calcular $\hat{e}(e,i)$ mediante la ecuación (5.15)
        para $j = 1 \cdots J$ hacer
          Actualizar $Q(e,i,j)$ de acuerdo a (5.6)
        fin_para
        Actualizar $T(e,i)$ de acuerdo a (5.7)
      fin_para
    fin_para
    Calcular $\tilde{e}_1^I = \arg\max_{e_1^I \in \{H(e_I): e_I \in \mathcal{E}\}} \{V(f_1^J \mid e_1^I; m)\}$
  fin_mientras
  devolver: $\tilde{e}_1^I$

Algoritmo 5.3: Versión por Viterbi del algoritmo iterativo de búsqueda para los modelos 3, 4 y 5.

los modelos 3 y 4. Dichos propuestas plantean una solución óptima al problema de modo que el coste de dichos algoritmos es exponencial.

En el apéndice B podemos ver como definir criterios de búsqueda para los modelos 3 y 4, y en [Mansilla 02] una implementación de ellos junto con toda la experimentación asociada para su estudio con respecto a eficiencia y eficacia en la traducción.

5.6 Complejidad computacional

La complejidad por iteración del algoritmo *IterativeDPSearchM2* (algoritmo 5.2), o lo que es lo mismo la del algoritmo 5.1 vienen marcadas por el par de bucles anidados para i y para_todo e. Dentro de estos bucles tenemos:

- El cálculo de $\hat{e}(e,i)$, que con conlleva un coste del orden de $O(\mid \mathcal{E} \mid \cdot J)$.
- El bucle para j, que conlleva un coste de $O(J)$, puesto que la actualización de $Q(e,i,j)$ tiene un orden constante.
- La actualización de $T(e,i)$ que también tiene coste constante:

Por lo tanto el coste computacional del algoritmo será del orden de:

$$O(J \cdot I_{max} \cdot L \cdot \mid \mathcal{E} \mid^2),$$

donde I_{max} es la longitud máxima de salida y L es el número de longitudes de salida testeadas.

A pesar de que la talla del problema viene dada por la variable J (longitud de la cadena a traducir), el factor realmente predominante es la talla del vocabulario de salida al cuadrado. A la vista de esta expresión ya podemos hacernos una idea de que tipo de tareas podrán abordarse con este algoritmo, es decir, aquellas en las que el vocabulario del lenguaje de salida se limite a varios miles de palabras (por ejemplo la tarea EuTrans-I). Si pretendemos utilizar este algoritmo en tareas que conlleven vocabularios del orden de decenas o cientos de miles de palabras (por ejemplo las tareas Verbmobil o Hansards) deberemos aplicar específicas optimizaciones heurísticas para poder abordarlas.

En la siguiente sección veremos que tipo de optimizaciones se pueden realizar para mejorar la eficiencia de estos algoritmos[3].

5.7 Optimizaciones en el proceso de búsqueda

De acuerdo a la complejidad computacional asociada a los algoritmos propuestos en las secciones anteriores parece razonable realizar un estudio de las posibles optimizaciones a utilizar para reducir el espacio de búsqueda requerido por dichos algoritmos. A continuación se proponen un par de optimizaciones, una que reduce la complejidad teórica del algoritmo y otra que reduce el espacio de búsqueda.

La propia naturaleza heurística de los algoritmos propuestos provoca que no podamos garantizar obtener la traducción "óptima" de una frase dada, o lo que es lo mismo, que los algoritmos no estarán libres de cometer errores de búsqueda[4]. A este tipo de errores les llamaremos errores inherentes al algoritmo.

Evidentemente, las optimizaciones que se proponen a continuación llevarán asociados más errores de búsqueda, con lo que deberemos establecer un compromiso entre eficiencia y calidad en la traducción.

[3] A partir de ahora, y en el resto de capítulos, hablaremos de optimizaciones a los algoritmos en el sentido de mejorar la eficiencia de los mismos. Desde el punto de vista de la optimalidad de la solución, se trataría por tanto de *restricciones*, pero hemos utilizado el término *optimizaciones* dado que no cabe duda que con ellas se mejoran las prestaciones de dichos algoritmos en cuanto a eficiencia se refiere

[4] Según se definieron en el capitulo 2

5.7.1 Reducción de la complejidad del algoritmo

Como ya hemos comentado el factor determinante de la complejidad de estos algoritmos es la talla del vocabulario de salida, lo cual hará prohibitivo el coste del algoritmo cuando intentemos abordar tareas con tallas de vocabularios grandes. Parece razonable no tener que iterar exhaustivamente los bucles `para_todo` $e \in \mathcal{E}$ (al igual que en el cálculo de la expresión (5.15)), que aparecen en los algoritmos. De hecho, bastaría simplemente con considerar aquellas palabras e que sean buenas traducciones de las palabras que conforman la frase de entrada f_1^J, con lo que la complejidad de los algoritmos se vería reducida considerablemente.

Esta optimización consiste en elegir las W mejores traducciones de cada palabra de la frase de entrada, y al conjunto de todas ellas le llamaremos $\mathcal{W}(f_1^J)$. Para ello haremos uso de la definición de *traducciones inversas* [Al-Onaizan et al. 99]. La probabilidad inversa $t^{-1}(e \mid f)$ de la traducción de una palabra origen f puede calcularse como:

$$t^{-1}(e \mid f) = \frac{t(f \mid e) \cdot p(e)}{\sum_{e'} t(f \mid e') \cdot p(e')} \ , \tag{5.17}$$

donde $p(e)$ es la probabilidad del unigrama e, la cual se puede estimar a partir del mismo corpus de entrenamiento utilizado para estimar $t(f \mid e)$ o, si es posible, a partir uno de mayor tamaño, dado que, siempre es más sencillo encontrar corpus monolingües que bilingües.

Esta optimización es fundamental para mantener bajo control la complejidad del algoritmo.

Si e_1^I es la traducción óptima de la frase f_1^J, entonces, toda palabra destino e_i deberá pertenecer al conjunto $\mathcal{W}(f_1^J)$, de lo contrario, nunca se podrá alcanzar la traducción óptima. En definitiva, esta optimización puede dar lugar a errores de búsqueda y la solución a este problema pasa por aumentar el valor de W.

Por otra parte, además de considerar las W mejores traducciones de cada palabra de la frase de entrada, también deberemos tener en cuenta aquellas posibles palabras del vocabulario de salida que no son traducción de ninguna palabra de la frase de entrada, es decir, las palabras de fertilidad cero. Al número de palabras de fertilidad cero a considerar le llamaremos Z y al conjunto de ellas le llamaremos $\mathcal{Z}$. La elección de este conjunto de palabras se puede realizar de dos formas distintas:

- Haciendo uso de los parámetros de fertilidad del modelo 3 (o superiores). En este caso solo tendremos que elegir las Z palabras e, que tengan mayor probabilidad $n(0 \mid e)$ asociada.
- Siendo puristas y teniendo en cuenta que el algoritmo *IterativeDPSearchM2* se basa única y exclusivamente en el modelo 2, no parece razonable hacer uso de la información de los modelos superiores para optimizarlo, por tanto una segunda posibilidad de construir el conjunto $\mathcal{Z}$ sería utilizando la información del alineamiento por Viterbi tras el entrenamiento del modelo

2. La elección consistirá en elegir las Z palabras que más veces se han quedado sin alinear en el alineamiento por Viterbi (para más información ver [García-Varea and Casacuberta 01]).

Un efecto similar al caso anterior ocurre con las palabras de fertilidad cero y el parámetro Z. De la misma forma, habrá hipótesis inalcanzables por el algoritmo en caso de que falte la palabra con fertilidad cero adecuada. Es decir, esta optimización también podrá provocar errores de búsqueda. La solución a este problema pasa por aumentar el valor de Z.

Teniendo esto en cuenta, la complejidad computacional pasará a ser:

$$O(J \cdot I_{max} \cdot L \cdot (JW + Z)^2) = O(J^3 \cdot I_{max} \cdot L \cdot W^2),$$

dado que en general JW será mayor que Z.

A pesar de que el coste computacional es proporcional a J^3, en la sección de experimentación veremos que se obtienen unos tiempos de respuesta aceptables.

Los valores de W y Z se establecen empíricamente, de modo que se establezca un compromiso entre eficiencia y calidad. En la sección siguiente se hace un estudio detallado de la influencia de estos parámetros.

En el algoritmo 5.4 podemos ver cómo quedaría el algoritmo *IterativeDP-SearchM2* teniendo en cuenta estas optimizaciones, donde con "⇐" hemos marcado las diferencias con respecto a la versión sin optimizaciones.

5.7.2 Reducción del espacio de búsqueda

En esta sección veremos una serie de optimizaciones heurísticas que darán lugar a realizar una búsqueda más eficiente, pero por contra todas ellas llevarán asociados errores de búsqueda.

Reducción del tamaño del trellis

Para reducir el espacio de búsqueda normalmente lo que se utiliza es la técnica conocida como *búsqueda en haz*[5] [Ney et al. 87], la cual se usa en la gran mayoría de los sistemas de reconocimiento del habla, así como en gran parte de problemas que se resuelven con técnicas de programación dinámica.

Esta técnica consiste en establecer un valor a una variable, que llamaremos *beam*, de modo que en cada etapa del proceso de búsqueda se eliminarán (o lo que es lo mismo no se considerarán en la próxima etapa) aquellos estados cuya *puntuación* parcial supere el umbral establecido por el *beam* con respecto a la mejor hipótesis computada hasta esa etapa. Esta técnica no involucra ningún coste adicional al algoritmo debido a que:

- La mejor hipótesis en una determinada etapa se determina en la propia iteración del bucle `para_todo` $e \in \mathcal{E}$, almacenándose en una variable temporal. Esta operación tendrá un coste constante.

[5] En inglés *Beam Search*

Algoritmo: ***IterativeDPSearchM2_opt_W+Z***
entrada: $I, t(\cdot), a(\cdot), f_1^J$
salida: $\hat{e}_1^I$
  //Inicialización
  Sea $\tilde{e}_1^J$ la solución al algoritmo ***DPSearchM2***
  //Iteración
  mientras (convergencia) hacer
    ${e'}_1^I = \tilde{e}_1^J$ {Actualización de hipótesis previa}
    para $i = 1 \cdots I$ hacer
      para $j = 1 \cdots J$ hacer
        $R(j,i) = t(f_j \mid e'_i) \cdot a(i \mid j, J, I)$ {Actualización de la matriz R}
      fin_para
    fin_para
    //Método
    para $i = 1 \cdots I$ hacer
      para_todo $e \in \{\mathcal{W}(f_1^J) \cup \mathcal{Z}\}$ hacer $\{\Leftarrow\}$
        Calcular $\hat{e}(e,i)$ mediante la ecuación (5.15)
        para $j = 1 \cdots J$ hacer
          Actualizar $Q(e,i,j)$ de acuerdo a (5.6)
        fin_para
        Actualizar $T(e,i)$ de acuerdo a (5.7)
      fin_para
    fin_para
    $\tilde{e}_1^I = H(\hat{e}_I)$, t.q.: $\hat{e}_I = \arg\max_{e_I \in \mathcal{E}} \left\{ T(e_I, I) \prod_{j=1}^{J} Q(e_I, I, j) \right\}$
  fin_mientras
  devolver: $\tilde{e}_1^I$

Algoritmo 5.4: Algoritmo iterativo de búsqueda para el modelo 2 con las optimizaciones W y Z.

- No es necesario ordenar los estados de acuerdo a su *puntuación*, pues una vez conocida la *puntuación* de la mejor hipótesis en la etapa i, para generar los estados de la etapa $i+1$, sólo se considerarán estados válidos aquellos que cumplan la condición del *beam*.

Hay que tener en cuenta que el valor del *beam* será determinante en la eficiencia de los algoritmos, hasta el punto en que un valor de $beam = 0$ solo permitirá la existencia de una hipótesis en cada etapa del proceso de búsqueda (la mejor hasta ese momento), con lo que el algoritmo pasaría a ser un algoritmo voraz.

En el algoritmo 5.5 podemos ver cómo quedaría el algoritmo *IterativeDP-SearchM2* teniendo en cuenta estas optimizaciones, donde con el símbolo "$\Leftarrow$" hemos marcado las partes que se ven afectadas por la modificación.

Algoritmo: ***IterativeDPSearchM2_opt_W+Z+beam***
entrada: $I, t(\cdot), a(\cdot), f_1^J, beam$
salida: $\hat{e}_1^I$
  //Inicialización
  Sea $\tilde{e}_1^J$ la solución al algoritmo ***DPSearchM2***
  //Iteración
  mientras (convergencia) hacer
    $e'^I_1 = \tilde{e}_1^J$ {Actualización de hipótesis previa}
    para $i = 1 \cdots I$ hacer
      para $j = 1 \cdots J$ hacer
        $R(j,i) = t(f_j \mid e'_i) \cdot a(i \mid j, J, I)$ {Actualización de la matriz R}
      fin_para
    fin_para
    //Método
    $mejorPuntuacion(0) = 0$
    para $i = 1 \cdots I$ hacer
      para_todo $e \in \{\mathcal{W}(f_1^J) \cup \mathcal{Z}\}$ hacer
        si $puntuacion(Estado(e,i)) \geq$
          $mejorPuntuacion(i-1) - beam$ entonces $\{\Leftarrow\}$
          Calcular $\hat{e}(e,i)$ mediante la ecuación (5.15)
          para $j = 1 \cdots J$ hacer
            Actualizar $Q(e,i,j)$ de acuerdo a (5.6)
          fin_para
          Actualizar $T(e,i)$ de acuerdo a (5.7)
          $mejorPuntuacion(i) = \max(mejorPuntuacion(i),$
                                  $puntuacion(Estado(e,i)))\{\Leftarrow\}$
        fin_si
      fin_para
    fin_para
    $\tilde{e}_1^I = H(\hat{e}_I)$, t.q.: $\hat{e}_I = \arg\max_{e_I \in \mathcal{E}} \left\{ T(e_I, I) \prod_{j=1}^{J} Q(e_I, I, j) \right\}$
  fin_mientras
  devolver: $\tilde{e}_1^I$

Algoritmo 5.5: Algoritmo iterativo de búsqueda para el modelo 2 con las optimizaciones W, Z y $beam$.

Reducción el número de longitudes de salida a testear

Como comentamos al final de la sección 5.2.1, dado que la longitud de la cadena de salida no es conocida a priori, debemos resolver el problema planteado en la ecuación (5.2) para las distintas posibles longitudes de salida.

Realizar esto para toda posible longitud de salida no solo es impracticable sino poco razonable, por tanto reduciremos sustancialmente el espacio total de

búsqueda estableciendo un rango aceptable de longitudes de salida a testear. Para obtener este rango haremos uso de la distribución de longitudes $\epsilon(I \mid J)$, en nuestro caso aproximado por una guassiana, de modo que la media de dicha gaussiana será la longitud media observada para todas las frases de entrada de longitud J. A partir de esta longitud media de salida, llamémosle $\bar{I}$, establecemos un rango de longitudes a testear de L longitudes a la izquierda y a la derecha de $\bar{I}$.

Con esta optimización reduciremos el espacio de búsqueda de longitudes de salida a testear de I_{max} a $2L + 1$.

Normalmente el valor de estos parámetros (*beam* y L) se establecen empíricamente, de modo que se establezca un compromiso entre eficiencia y calidad. En la sección siguiente se hace un estudio detallado de la influencia de estos parámetros.

Estas optimizaciones también pueden provocar errores de búsqueda. La solución pasa una vez más por aumentar dichos valores.

Recombinación de hipótesis

Según hemos planteado el criterio de búsqueda para el algoritmo *DPSearchM2* (ver sección 5.2), es válido siempre y cuando utilicemos modelos de bigramas en la búsqueda. Normalmente, y dado su potencial predictivo, en la experimentación utilizaremos trigramas. Esto conllevará redefinir ligeramente el algoritmo, de modo que los estados del trellis que definen unívocamente una hipótesis deberán contener dos palabras destino (e', e, i), es decir, aquellas que formen parte de la historia del modelo de trigramas que estemos utilizando. En general para un modelo de n-gramas necesitaremos las $n - 1$ palabras de la historia para definir el estado. De este modo, tal y como se planteó el algoritmo *DPSearchM2*, si utilizamos trigramas en el proceso de búsqueda, se estará realizando una recombinación de hipótesis durante el proceso. Es decir, se recombinarán aquellas hipótesis (o estados) que contengan la misma palabra final de la historia del modelo de lenguaje, es decir, el estado (e, i) resumirá la información de los estados $(e', e, i), \forall e' \in \mathcal{E}$, y la *puntuación* asociada a dicho estado será:

$$puntuacion(e, i) = \max_{e'}\{puntuacion(e', e, i)\}$$

El efecto de esto se traduce en poder realizar una búsqueda mucho más rápida, pero ineludiblemente también a la introducción de errores de búsqueda. En la sección siguiente veremos experimentalmente el efecto de la recombinación en cuanto a eficiencia y calidad en la traducción.

5.8 Discusión de los algoritmos basados en PD

En esta sección vamos a discutir experimentalmente las cualidades de los algoritmos comentados en apartados anteriores. Para ello llevaremos a cabo

una serie de experimentos utilizando la tarea EuTrans-I dada su sencillez. Concretamente hemos escogido las 299 frases del conjunto test mostrado en la tabla 2.2 (ver sección 2.7.1, página 46) de longitud 9. La elección de este subconjunto no ha sido otro que el escoger las frases más cercanas a la longitud media de las frases de test, siendo ésta aproximadamente 9. Otro motivo del porqué de elegir este subconjunto es el de que los resultados aquí mostrados sean comparables al resto de algoritmos de búsqueda presentados en capítulos posteriores, siendo los algoritmos de pila los que establecen una restricción en cuanto a la longitud máxima de frases de entrada factible de ser traducida en un tiempo razonable. A titulo informativo, en la tabla 5.1 podemos ver las características de este corpus de test, y podemos resaltar que la perplejidad de un modelo de lenguaje de trigramas es igual que la de la tarea total.

		Castellano	Inglés
Test	Frases	299	
	Palabras	2 691	2 879
	Perplejidad (Trigramas)	–	3.5

Tabla 5.1: Estadísticas del corpus utilizado en el establecimiento de los parámetros de los algoritmos de búsqueda.

Los modelos de traducción, que utilizaremos en toda la experimentación que mostraremos a continuación, se han entrenado con el conjunto de entrenamiento mostrado en la tabla 2.2, es decir un corpus de 10 000 pares de frases, utilizando el esquema de entrenamiento $1^5 2^5 3^5 4^5 5^5$. Por otra parte, con respecto a los modelos de lenguaje, mientras no digamos lo contrario, supondremos que estaremos hablando de modelos de trigramas. En concreto, en la implementación de estos algoritmos hemos utilizado modelos de trigramas suavizados mediante la técnica de descuento *Good Turing* [Chen and Goodman 96], y han sido entrenados con la herramienta SRILM [Stolcke 97]. Del mismo modo, dichos modelos de lenguaje han sido entrenados con las correspondientes 10,000 frases para el para el lenguaje de salida.

La forma de hacer el estudio planteado tiene tres etapas:

1. Establecer un algoritmo básico sobre el que se realizarán las pruebas.
2. Hacer un recorrido por las distintas optimizaciones planteadas, experimentando sobre su efecto en la eficiencia y la tasa de aciertos del algoritmo.
3. Por último, haremos una comparativa sobre la influencia que tienen en la eficiencia, el algoritmo que se escoja, el modelo de traducción y el tamaño de frase.

Como resultado de este estudio, escogeremos un traductor óptimo con el que obtendremos más resultados en la sección siguiente.

5.8.1 Establecimiento de los parámetros de los algoritmos basados en PD.

Vamos a fijar por defecto (y en base a experimentaciones preliminares) las características básicas del traductor de la siguiente manera:

- Algoritmo elegido: *IterativeDPSearchM2_opt_W+Z+beam*
- Inicialización heurística: Si
- Parámetros: $W = 12$, $Z = 24$, $L = 4$, y $beam = 5000$
- Número de iteraciones: 3
- Recombinación: No

En lo sucesivo cuando hablemos de aciertos en la traducción estaremos haciendo referencia a valor complementario de **SER**, es decir lo que sería el **SAR** (*sentence accurate rate*), de este modo cuando se muestre el número de aciertos, no se mostrará el índice **SER**.

Es interesante conocer cual sería el mínimo número de errores de búsqueda que podamos cometer dadas unas condiciones/valores óptimos de los parámetros de optimización. Para ello hemos implementado una versión del algoritmo *IterativeDPSearchM2* en el que se realiza una búsqueda guiada estableciendo el valor de los parámetros W y Z a su valor óptimo teóricamente, es decir, que contengan solamente aquellas palabras que en realidad deberán formar parte de la traducción. Esto es posible llevarlo a cabo dado que conocemos, para cada frase de entrada, su frase de referencia. En la tabla 5.2 podemos ver la información del modelo de error y las tasas de error que se podrían garantizar. A la vista de esos resultados podemos adelantar que el número de errores inherentes al algoritmo *IterativeDPSearchM2* es igual al 1.67%.

ErrBusq	**ErrModel**	**Aciertos (SAR)**	**WER**	**PER**
1.67	16.72	81.61	3.61	2.47

Tabla 5.2: Errores inherentes al algoritmo *IterativeDPSearchM2*, para la tarea EuTrans-I.

Cabe matizar, que en las tablas relativas a los experimentos que expondremos a continuación se incluye:

- El tiempo medio de ejecución en segundos (**secs**) por frase[6].
- El modelo de error, es decir: porcentaje errores de búsqueda (**ErrBusq**), porcentaje errores del modelo (**ErrModel**) y porcentaje de aciertos (**Aciertos**).

[6] Todos los experimentos presentados en esta parte de la memoria han sido realizados en una máquina Pentium III (Coppermine) a 800 MHz con 2.0 G de memoria

- Por último, las tasas de error (WER y PER), en tanto por ciento.

Influencia del número de traducciones inversas (parámetro W)

Cómo podemos ver en la tabla 5.3, a medida que aumenta el valor de W (número de traducciones inversas para cada palabra f de la frase de entrada) aumenta la calidad de la traducción y del mismo modo el tiempo medio por frase. A la vista de los resultados podemos establecer $W = 20$ como valor aceptable, pues a partir de él no se observa una mejora sustancial y si aumenta el tiempo de ejecución.

W	secs	ErrBusq	ErrModel	Aciertos	WER	PER
5	7.25	12.37	45.48	42.14	12.19	10.32
10	11.11	7.69	49.83	42.47	11.81	9.76
15	13.95	7.36	50.17	42.47	11.81	9.76
20	16.34	7.36	50.17	42.47	11.88	9.83
25	17.22	7.36	50.17	42.47	11.88	9.83
30	18.26	7.36	50.17	42.47	11.88	9.83
35	19.04	7.36	50.17	42.47	11.88	9.83
40	19.67	7.36	50.50	42.14	11.91	9.86
45	20.23	7.36	50.50	42.14	11.91	9.86
50	20.95	7.36	50.50	42.14	11.91	9.86

Tabla 5.3: Influencia del número de traducciones inversas para una palabra origen dada (parámetro W).

Influencia del número de palabras de fertilidad cero (parámetro Z)

Recordamos que el parámetro Z indica el número de palabras de fertilidad cero a utilizar. A la vista de los resultados de la tabla 5.4, en este caso cabe hacer las mismas consideraciones que para el parámetro W. Podemos ver que a partir de un valor $Z = 30$ no se obtiene mejoras en la traducción y si en cambio mayor tiempo de ejecución.

Influencia del valor del parámetro *beam*

El valor de este parámetro es determinante para la eficiencia del algoritmo. Un valor de $beam = 0$ hace al algoritmo tremendamente rápido pero a su vez muy malo en lo que se refiere a calidad en la traducción. A la vista de la tabla 5.5 podemos ver que a partir de un valor de $beam = 20$ no se obtienen mejoras sustanciales, y si se deprecian las cualidades el algoritmo. De hecho, y dado que estamos trabajando con logaritmos, un valor de 20 ya permite una exploración bastante exhaustiva del espacio de búsqueda.

Z	segs	ErrBusq	ErrModel	Aciertos	WER	PER
5	6.41	36.12	33.44	30.43	16.26	13.93
10	7.44	29.10	37.79	33.11	14.66	12.26
15	8.83	19.40	44.82	35.79	13.75	11.32
20	11.28	18.39	45.48	36.12	13.55	11.22
25	12.61	7.36	50.17	42.47	11.81	9.76
30	14.20	4.68	51.51	43.81	11.43	9.55
35	16.10	4.68	51.51	43.81	11.43	9.55
40	18.10	5.02	51.51	43.48	11.46	9.59
45	20.76	5.02	52.51	42.47	11.95	9.93
50	23.19	5.02	52.51	42.47	11.95	9.93

Tabla 5.4: Influencia del número de palabras de fertilidad cero consideradas (parámetro Z).

beam	segs	ErrBusq	ErrModel	Aciertos	WER	PER
0	2.89	8.70	48.16	43.14	12.40	10.07
5	4.70	7.69	48.49	43.81	11.74	9.66
10	6.74	7.36	49.83	42.81	11.77	9.76
15	8.53	7.36	50.17	42.47	11.81	9.76
20	9.66	7.36	50.17	42.47	11.81	9.76
25	10.69	7.36	50.17	42.47	11.81	9.76
30	11.50	7.36	50.17	42.47	11.81	9.76
35	12.00	7.36	50.17	42.47	11.81	9.76
40	12.34	7.36	50.17	42.47	11.81	9.76
50	12.51	7.36	50.17	42.47	11.81	9.76
100	12.51	7.36	50.17	42.47	11.81	9.76
500	12.51	7.36	50.17	42.47	11.81	9.76

Tabla 5.5: Influencia de la poda basada en la búsqueda en haz (parámetro *beam*).

Influencia del rango de longitudes de salida (parámetro L)

Este parámetro, que establece el número de longitudes de salida a testear, no influye tanto como los anteriores en lo que se refiere a eficacia del algoritmo, aunque si en cuanto a eficiencia. A la vista de la tabla 5.6 podemos ver que un valor de $L = 3$ es más que suficiente para garantizar cierta calidad en la traducción.

L	secs	ErrBusq	ErrModel	Aciertos	WER	PER
1	4.16	27.42	36.79	35.79	13.96	12.26
2	6.94	10.03	48.83	41.14	12.05	10.14
3	9.72	7.69	50.17	42.14	11.88	9.83
4	12.51	7.36	50.17	42.47	11.81	9.76
5	15.34	7.36	50.17	42.47	11.81	9.76
6	18.13	7.36	50.17	42.47	11.81	9.76

Tabla 5.6: Influencia del número de longitudes de salida a testear (parámetro L)

5.8.2 Estudio empírico de eficiencia

Para la experimentación que llevaremos a cabo a continuación utilizaremos los mismos valores de los parámetros que los utilizados hasta ahora.

Influencia del número de iteraciones del algoritmo

En la tabla 5.7 podemos ver la calidad de la traducción en cada iteración del algoritmo *IterativeDPSearchM2*. Podemos ver que el algoritmo converge en la tercera iteración y que la ganancia en cuanto a calidad en traducción es sustanciosa de la primera iteración (o lo que es lo mismo utilizando el algoritmo *DPSearchM2*) a la tercera. Lógicamente, el tiempo de traducción medio se ve incrementado a medida que aumentamos el número de iteraciones. De estos resultados podemos concluir que realizar tres iteraciones del algoritmo es más que suficiente.

It.	secs	WER	PER
1	4.17	15.21	12.30
2	8.37	11.95	9.66
3	12.50	11.80	9.76
4	16.67	11.80	9.76
5	20.84	11.80	9.76

Tabla 5.7: Influencia del número de iteraciones en el algoritmo *IterativeDP-SearchM2* utilizando inicialización heurística.

Influencia de la inicialización heurística

La inicialización heurística del valor del array R es determinante en cuanto al número de iteraciones a realizar por el algoritmo. En la tabla 5.8 se muestran las tasas de error en cada iteración del algoritmo *IterativeDPSearchM2* sin utilizar la

inicialización heurística, la cual es directamente comparable con la tabla 5.7 en la que se muestra el mismo experimento pero utilizando dicha inicialización. Observando estos resultados se puede ver que utilizando la inicialización heurística bastaría con realizar dos iteraciones del algoritmo para obtener prácticamente los mismos resultados que con 3 o más iteraciones sin la inicialización. Esto evidentemente influye en la eficiencia de los algoritmos propuestos, viéndose reducido en un tercio el tiempo medio de traducción por frase, y lo que es más, también ayuda a obtener mejores tasas de error.

It.	secs	WER	PER
1	4.22	18.03	14.83
2	8.43	13.13	10.42
3	12.64	12.43	10.07
4	16.85	12.43	10.07
5	21.06	12.43	10.07

Tabla 5.8: Influencia del número de iteraciones en el algoritmo *IterativeDP-SearchM2*, sin utilizar inicialización heurística.

Influencia del uso de la recombinación de hipótesis

Como era de esperar y a la vista de los resultados de la tabla 5.9 podemos ver como el uso de la recombinación deprecia las prestaciones del algoritmo pero a su vez reduce sustancialmente el tiempo de ejecución. La utilización de esta optimización solamente tendría sentido en un sistema de traducción que requiera una rápida respuesta en detrimento de una mayor calidad.

Rec.	secs	ErrBusq	ErrModel	Aciertos	WER	PER
Si	3.23	13.71	46.15	40.13	13.34	10.56
No	12.51	7.36	50.17	42.47	11.81	9.76

Tabla 5.9: Influencia del uso de la recombinación de hipótesis.

Influencia del modelo en la versión por Viterbi

En la tabla 5.10 podemos ver los resultados de traducción para la familia de algoritmos *IterativeDPSearchVitMn*, obteniéndose los mejores resultados para el modelo 4, pero a su vez algo más lento que para los modelos inferiores. En la tabla se ha añadido la entrada modelo 0, que hace referencia al algoritmo *IterativeDPSearchM2*, para hacer más fácil la comparación entre ellos.

A la vista de estos resultados podemos ver que solamente se obtienen mejores resultados cuando se utilizan los modelos 4 y 5, es decir ni el modelo 3 ni el HMM aportan información que ayude a obtener mejores hipótesis entre iteraciones. Del mismo podemos apreciar que conforme aumenta la complejidad del modelo se reducen el número de errores del modelo, repartiéndose en parecidas proporciones entre errores de búsqueda y aciertos. También podemos ver que la versión por Viterbi para el modelo 2 es prácticamente idéntica a versión convencional.

De estos resultados también podemos hacernos una idea de la sobrecarga que supone el cálculo del alineamiento por Viterbi. En cuanto a los modelos 1, 2 y 3, el incremento es prácticamente despreciable, en cambio no lo es para los modelos que incluyen dependencias de primer orden, como lo son los modelos HMM, 4, y 5.

Mod.	secs	ErrBusq	ErrModel	Aciertos	WER	PER
0	12.51	7.36	50.17	42.47	11.81	9.76
1	12.67	10.70	47.49	41.81	12.30	10.18
2	12.74	7.69	48.83	43.48	11.95	9.86
H	15.07	10.70	46.82	42.47	12.37	10.28
3	13.40	10.37	46.15	43.48	12.16	10.18
4	16.93	13.04	40.80	46.15	10.35	9.66
5	31.58	11.37	41.47	47.16	10.63	9.31

Tabla 5.10: Influencia del modelo en la versión por Viterbi.

5.8.3 Elección del algoritmo

Vistos los resultados obtenidos en la experimentación llevada a cabo en esta sección, e intentando establecer un compromiso entre eficiencia y aciertos en la traducción, podemos concluir que una buena elección de un traductor será aquel que: utilice el algoritmo *IterativeDPSearchVitM4*, con inicialización heurística y sin recombinación de hipótesis. Como ya hemos comentado, será importante escoger un buen valor para los parámetros de optimización, concretamente hemos escogido los valores de $W = 30$, $Z = 30$, $It = 3$, $L = 4$, y $beam = 50$.

5.9 Resultados de traducción

Vamos a llevar a cabo una serie de experimentos utilizando un traductor con los parámetros y características establecidos en la sección anterior. Para hacer dichos experimentos se han escogido dos tareas distintas: EuTrans-I, y Hansards, cuyas características fueron descritas en el capítulo 2, sección 2.7.

Concretamente utilizaremos los siguientes conjuntos de test: 2996 frases para la tarea EuTrans-I, y 500 frases para la tarea Hansards, de longitudes que varían de 4 a 12 palabras por frase.

5.9.1 Resultados con la tarea EuTrans-I

De acuerdo a la experimentación previa, los parámetros de optimización que utilizaremos para este experimento son: $W = 30$, $Z = 30$, $It = 3$, $L = 4$, y $beam = 50$. Hemos realizado dos experimentos con esta tarea, uno utilizando el *IterativeDPSearchVitM4* (fila correspondiente a **Mod. 4**) y otro utilizando el algoritmo *IterativeDPSearchM2* (fila correspondiente a **Mod. 0**)

En la tabla 5.11 podemos ver los resultados de traducción para las 2996 frases de test de la tarea EuTrans-I, para ambos algoritmos, utilizando y sin utilizar recombinación de hipótesis.

Rec	Mod.	secs	ErrBus	ErrModel	Aciertos	WER	PER
Si	**0**	17.80	27.74	44.39	27.87	16.67	12.92
No	**0**	80.44	5.64	57.74	36.62	13.39	10.72
Si	**4**	22.58	39.29	31.21	29.51	14.38	11.95
No	**4**	99.33	14.52	40.32	45.16	10.25	9.27

Tabla 5.11: Resultados con la tarea EuTrans-I para los algoritmos basados en PD.

En la tabla 5.12 podemos ver resultados de traducción para un subconjunto de esta tarea, que contiene las frases de entrada de longitud menor e igual que 15. Estos resultados se muestran para poder compararlos con el resto de algoritmos de búsqueda, dado que estos tienen serias limitaciones con respecto al tamaño de frase. En la misma tabla podemos ver una vez más el efecto de la recombinación, siendo esta optimización muy efectiva en cuanto a eficiencia se refiere pero a su vez introduce gran cantidad de errores de búsqueda, lo que se traduce en una depreciación significativa en la eficacia del algoritmo. También se presentan resultados con el la versión por Viterbi para el modelo 4 por ser el que obtiene los mejores resultados.

5.9.2 Resultados con la tarea Hansards

Para la tarea de Hansards hemos realizado experimentos para cinco corpus de 100 frases cada uno, de frases de longitud 4, 6, 8, 10 y 12, dada la complejidad que esta tarea conlleva, en principio para obtener resultados comparables con el resto de algoritmos de búsqueda.

Los modelos de traducción utilizados para esta tarea se han entrenado utilizando el corpus mostrado en la tabla 2.4 consistente en 128,000 pares frases de entrenamiento, siguiendo el esquema de entrenamiento $1^5 2^5 3^5 4^5 5^5$. En cuanto

Rec.	Mod.	secs	ErrBus	ErrModel	Aciertos	WER	PER
Si	**0**	10.03	28.98	40.82	30.20	16.25	12.86
No	**0**	55.70	5.50	55.24	39.26	12.71	10.50
Si	**4**	25.59	18.93	40.52	40.55	11.06	9.95
No	**4**	69.46	12.18	45.11	42.72	10.19	9.44

Tabla 5.12: Resultados con la tarea EuTrans-I, para frases de longitud menor e igual que 15.

al modelo de lenguaje se han utilizado trigramas con la técnica de descuento *Good Turing* y ha sido entrenado con las correspondientes 128,000 frases para el lenguaje de salida.

De acuerdo a la experimentación previa (ver apéndice C), los parámetros de optimización que utilizaremos para este experimento son: $W = 30$, $Z = 30$, $It = 3$, $L = 4$, y $beam = 50$; y el algoritmo utilizado será el *IterativeDPSearchM2* con recombinación. El uso de la recombinación en esta tarea es crucial pues de lo contrario nos iríamos a tiempos de ejecución impracticables.

En la tabla 5.13 podemos ver los resultados de traducción para las 500 frases de test de la tarea HANSARDS, distinguiendo entre las frases de distinta longitud. Dada la complejidad de la tarea, y en similitud con otros trabajos publicados utilizando esta tarea, podemos decir que estos algoritmos se comportan bien en cuanto a las tasas de error, pero vemos como aumenta considerablemente el tiempo de ejecución conforme aumenta la longitud de la frase de entrada.

Long.	secs	ErrBus	ErrModel	Aciertos	WER	PER
4	11.70	1	43	56	30.17	30.17
6	30.88	6	84	10	50.59	48.90
8	68.93	2	86	12	52.39	48.54
10	143.22	2	96	2	56.06	50.62
12	259.62	2	97	1	63.75	55.69
media	102.87	3	81.20	16.20	50.53	46.78

Tabla 5.13: Resultados con la tarea Hansards para los algoritmos basados en PD.

En la tabla 5.14 podemos ver los resultados del mismo experimento de traducción pero sin realizar recombinación de hipótesis. Como se puede ver, el número de errores de búsqueda es 0 para todos los corpus, aunque a decir verdad la calidad de la traducción no sufre una mejora sustancial. Por contra, los tiempos de respuesta son tremendamente superiores. Estos resultados nos llevan a la conclusión de que para tareas complejas la recombinación se hace indispensable para poder obtener tiempos de traducción aceptables, sin que ello merme

la calidad de la traducción sustancialmente. De hecho los resultados para frases de longitud 12 no se muestran pues requieren unos tiempos extremadamente altos. Otra posibilidad para reducir los tiempo de ejecución pasaría por reducir drásticamente los valores de los parámetros W y Z.

Long.	secs	ErrBus	ErrModel	Aciertos	WER	PER
4	117.56	0	44	56	30.17	29.93
6	638.24	0	86	14	46.53	44.84
8	1970.33	0	87	13	52.39	48.40
10	4591.54	0	96	4	55.24	50.72

Tabla 5.14: Resultados con la tarea Hansards para los algoritmos basados en PD, utilizando recombinación de hipótesis.

5.10 Conclusiones

En este tema hemos expuesto una familia de algoritmos de búsqueda basados en programación dinámica siguiendo la filosofía del modelo 2, estableciendo los criterios de búsqueda para los distintos algoritmos.

Por otra parte hemos estudiado su complejidad teórica, y diversas formas de optimizar los algoritmos, realizando un estudio empírico de eficiencia para los distintos parámetros de optimización y las distintas versiones de los algoritmos.

De acuerdo con la experimentación llevada a cabo y a la vista de los resultados de traducción obtenidos en la sección anterior, podemos concluir que:

- Son susceptibles de ser utilizados en tareas de complejidad reducida.
- Estableciendo un compromiso entre eficiencia y calidad, se pueden obtener traducciones aceptables en unos tiempos de ejecución medios.
- Dada la naturaleza heurística de estos algoritmos no parece razonable que sean utilizados en sistemas de traducción que requieran de una calidad alta, en cambio, si en sistemas de ayuda a la traducción, en los que la calidad puede que no sea un parámetro tan relevante cómo la velocidad de traducción.
- Los resultados obtenidos son competitivos comparados con otras técnicas más complejas, como por ejemplo los publicados en [Och 02].

Como hemos visto en este capítulo, la gran desventaja del algoritmo *IterativeDPSearchM2*, y sus versiones, es el tener que repetir el proceso de búsqueda para el conjunto de longitudes de salida a testear. La idea que tenemos en mente para el futuro pasa por eliminar esta repetición intentando que el algoritmo se

adapte dinámicamente a las distintas longitudes. Esto provocará que el algoritmo sea incluso más heurístico de lo que lo es ahora. Investigaciones preliminares en este sentido nos revelan que la depreciación de las prestaciones del algoritmo no es despreciable, pero la ganancia en cuanto a tiempo de ejecución, es decir la reducción en el tiempo medio consumido por frase, es sustancial, lo cual es bastante alentador. Por ello, pensamos que investigar un poco más en esta dirección podría derivar en obtener resultados cuando menos interesantes.

Capítulo 6

Algoritmos de pila

OTRA aproximación para la resolución del problema de la búsqueda en traducción automática estadística son los algoritmos de pila, o *stack decoding* según el término utilizado comúnmente en la bibliografía sobre la materia, los cuales hacen uso del paradigma de ramificación y poda para la resolución de problemas.

Bajo determinadas condiciones, los algoritmos de pila permiten obtener la solución óptima al problema de la traducción automática. La bondad de dicha solución se mide a través de una *puntuación* o *score* asociada a cada hipótesis; la cual se construye computando todos y cada uno de los términos de una fórmula asociada a los modelos que se estén utilizando.

6.1 Fundamentos de los algoritmos de pila

El algoritmo *stack decoding* fue introducido inicialmente en el dominio del reconocimiento del habla por Jelinek en 1969 [Jelinek 69], su funcionamiento consiste en el desarrollo incremental de hipótesis parciales, cada una de las cuales tiene asignada una *puntuación* también parcial, que vendrá definido por la forma en que las palabras de la frase de entrada y las de la hipótesis se vayan alineando. Dichas hipótesis son almacenadas en una pila o cola de prioridades, de manera que quedan ordenadas por su *puntuación* dentro de la cola[1].

[1] A pesar de que a estos algoritmos se les llama de pila, normalmente en la implementación se utilizan colas de prioridades, por lo tanto en lo sucesivo haremos alusión indistintamente al término *pila* o *cola* para referirnos a la estructura de datos que almacena las hipótesis activas durante el proceso de búsqueda

El algoritmo *stack decoding* es un algoritmo que construye su solución iterativamente, extrayendo de la cima de la cola la hipótesis más prometedora (aquella con mayor *puntuación*) en cada iteración y sometiéndola a un proceso de *expansión*. El concepto de *expansión* de una hipótesis puede definirse, usando la terminología básica de los problemas de búsqueda, como la aplicación de todos los operadores posibles sobre esa hipótesis. Esto se puede ver resumido, y de forma algorítmica, en el algoritmo 6.1. En las siguientes secciones comentaremos más detalles acerca de él.

Algoritmo: *StackDecoding*
entrada: $Pila : P, f_1^J$
salida: $\hat{e}_1^I$
  $//Inicialización$
  $apilar(P, e_0)$ {apilar la hip. nula}
  $//Método$
  $hip = desapilar(P)$ {extraer la mejor hip. de la pila}
  repetir
    Expansión(h) {expandir la hipótesis}
    $hip = desapilar(P)$
  hasta hip.es_completa() {encontrar una hip. completa}
  devolver: $e_1^I \in hip$

Algoritmo 6.1: Algoritmo básico *StackDecoding*.

Como ya hemos comentado varias veces en esta tesis el problema de la búsqueda en la traducción automática estadística radica en resolver el problema de la maximización dada en la ecuación 1.2. En este caso, el criterio de búsqueda se puede definir de forma ligeramente diferente al criterio utilizado en el tema anterior, debido precisamente a la forma de generar la hipótesis de salida junto con el alineamiento asociado entre ella y la cadena de entrada. Por lo tanto, utilizando un modelo de alineamiento, la ecuación 1.2 se puede reescribir cómo:

$$\begin{aligned} \hat{\mathbf{e}} &= \arg\max_{\mathbf{e}} \{Pr(\mathbf{e}) \cdot Pr(\mathbf{f} \mid \mathbf{e})\} \\ &= \arg\max_{\mathbf{e}} \left\{ Pr(\mathbf{e}) \cdot \sum_{\mathbf{a}} Pr(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) \right\} \end{aligned} \tag{6.1}$$

y siguiendo la aproximación por maximización tendríamos:

$$\begin{aligned} \hat{\mathbf{e}} &\approx \arg\max_{\mathbf{e}} \left\{ Pr(\mathbf{e}) \cdot \max_{\mathbf{a}} Pr(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) \right\} \\ &= \arg\max_{\langle \mathbf{e}, \mathbf{a} \rangle} \{Pr(\mathbf{e}) \cdot Pr(\mathbf{f}, \mathbf{a} \mid \mathbf{e})\} \end{aligned} \tag{6.2}$$

Por tanto el criterio de búsqueda en este caso consiste en encontrar el par $\langle \mathbf{e}, \mathbf{a} \rangle$ que define a la hipótesis de salida. Por este motivo, el alineamiento $\mathbf{a}$ obtenido

se puede decir que será el alineamiento por Viterbi entre la frase de entrada y la hipótesis obtenida, siempre teniendo en cuenta las restricciones del modelo de alineamiento que estemos utilizando.

6.2 Algoritmo básico *StackDecoding*

Visto el funcionamiento general de los algoritmos de pila, estamos en disposición de establecer sus ingredientes básicos:

- Modelos estadísticos de traducción y de lenguaje.
- Operadores que permitan construir incrementalmente (expandir) las hipótesis y el correspondiente mecanismo de asignación de puntuaciones a las hipótesis parciales.
- Algoritmo de expansión de hipótesis.
- Estructura de datos asociada a una hipótesis parcial.
- Cola de prioridades cuyo criterio de ordenación sea la *puntuación* de las hipótesis.

Estos elementos habrán de ir diseñándose adecuadamente siguiendo un orden como el siguiente:

1. Elección de los modelos estadísticos.
2. Definición de los operadores que actúan sobre una hipótesis en base al modelo que a utilizar y a la forma en que éstos afectan a la *puntuación* de una hipótesis parcial.
3. Descripción del algoritmo de expansión, es decir cómo se combinan dichos operadores.
4. Determinar la estructura de datos de cada hipótesis en base a: los operadores, el mecanismo de asignación de puntuaciones y el algoritmo de expansión.

Veamos con detenimiento cada uno de ellos, en donde además se darán ciertos detalles relevantes para la implementación de estos algoritmos de búsqueda.

6.2.1 Elección del modelo estadístico

En el tema anterior hemos centrado el diseño de algoritmos de búsqueda basándonos en los modelos 1 y 2. Los algoritmos de pila nos proporcionan una atractiva característica que es la relativa sencillez con la que se pueden aplicar a modelos de traducción más complejos. Por ello, en este capítulo, desarrollaremos algoritmos de pila para los modelos 3 y 4.

Con respecto a los modelos de lenguaje, mientras no digamos lo contrario, supondremos que estaremos hablando de modelos de trigramas. En concreto, en la implementación de estos algoritmos hemos utilizado los mismos modelos de lenguaje que en el tema anterior, es decir, modelos de trigramas suavizados mediante la técnica de descuento *Good Turing*.

6.2.2 Operadores sobre una hipótesis y contribuciones a su *puntuación*.

En [Berger et al. 96a] se definen una serie de operadores para su utilización con el modelo 3. Dichos operadores producen *extensiones* en las hipótesis parciales, de modo que una *extensión* se produce teniendo en cuenta un y sólo un elemento de la frase origen. Existen dos tipos de extensiones: abiertas y cerradas. Una extensión se denomina *abierta* cuando la última palabra que forma la hipótesis parcial puede alinearse con nuevas palabras de la frase origen en posteriores extensiones. Por el contrario, se dice que una extensión es *cerrada* cuando la última palabra añadida ya no puede alinearse con otras palabras origen en futuras extensiones.

A continuación se muestra una lista con los operadores presentados en [Berger et al. 96a] y su descripción, los cuales también han sido utilizados en nuestra implementación:

$add(e, j)$**:** Se añade la palabra destino e a la hipótesis y se alinea con la palabra que ocupa la posición j de la frase origen.

$extend(j)$**:** Se alinea la última palabra destino con la palabra que ocupa la posición j.

$close()$**:** Se cierra una hipótesis abierta.

$addZFert(ze, e, j)$**:** Se añade la palabra destino ze a la que se asigna fertilidad cero, y a continuación se añade la palabra destino e, alineándose con la palabra origen en la posición j.

$addNull(j)$**:** La palabra origen en la posición j se alinea con la palabra vacía (e_o) de la estructura destino, o lo que es lo mismo, la palabra origen f_j será una palabra espuria.

Existen una serie de precondiciones aplicables a cada uno de los operadores que rigen la evolución del estado de una hipótesis parcial. En la figura 6.1 podemos ver el diagrama de estados que modela dichas precondiciones, y la transición entre ellos.

El cálculo de la *puntuación* para cada hipótesis parcial se obtiene multiplicando la *puntuación* actual por la *puntuación* de la extensión.

Conocido el modelo estadístico y definidos los operadores, queda por ver cómo se establecen las contribuciones de cada uno a la *puntuación* de las hipótesis

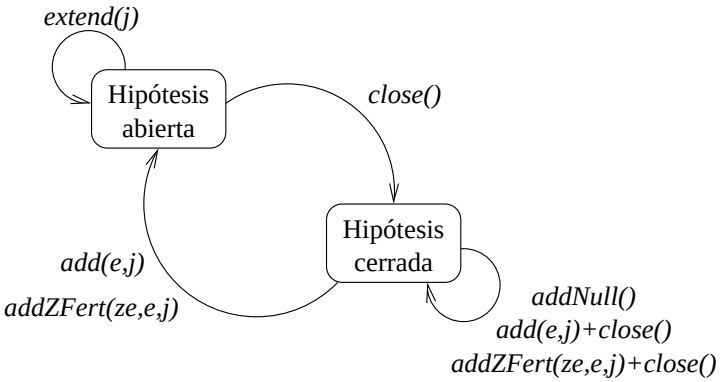


Figura 6.1: Diagrama de estados de hipótesis.

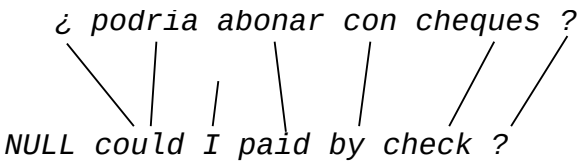


Figura 6.2: Traducción y alineamiento obtenidos mediante la aplicación de un algoritmo de pila.

parciales sobre las cuales han de aplicarse, de manera que se vaya construyendo incrementalmente la fórmula asociada al modelo. Para mostrar dichas contribuciones vamos a recurrir a un ejemplo tomado de la tarea EuTrans-I, concretamente veremos como traducir la frase:

¿ podría abonar con cheques ?

en su correspondiente traducción:

could I pay by check ?

El traductor llega hasta esta frase combinando de determinada forma los operadores anteriores, de manera que la hipótesis completa final tiene el alineamiento que se muestra en la figura 6.2. Dicho alineamiento se obtiene aplicando la siguiente secuencia de operadores:

$$add(\textsf{could}, 1) \rightarrow extend(2) \rightarrow close() \rightarrow addZFert(\textsf{I}, \textsf{pay}, 3) \rightarrow close() \rightarrow add(\textsf{by}, 4) \rightarrow close() \rightarrow add(\textsf{check}, 5) \rightarrow close() \rightarrow add(\textsf{?}, 6) \rightarrow close()$$

Veamos paso a paso cómo se construyó la hipótesis de salida. Antes de aplicar cualquier operador, se parte de una *hipótesis vacía*, una hipótesis vacía es aquella que no contiene ninguna palabra destino excepto e_0, y una *puntuación* igual a suponer que todas las palabras origen son no espurias. Lo que equivale a decir que se asume que inicialmente $\phi_0 = 0$.

Partiendo de una hipótesis vacía, veremos a continuación, una por una, las contribuciones de los distintos operadores, suponiendo que utilizamos el modelo 3 como modelo de traducción y trigramas como modelo de lenguaje.

Contribución del operador $add(\textit{could}, 1)$

La operación $add(\textit{could}, 1)$ añade la primera palabra (*could*) a la frase destino, en la figura 6.3 se muestra el alineamiento parcial como efecto de dicha operación. En cuanto a su contribución a la *puntuación*, éste queda como sigue:

$$p(\textit{could} \mid \$, \$) \times$$
$$2\left(\sum_{\phi=2..\phi_{max}} n(\phi \mid \textit{could})\right) \cdot t(? \mid \textit{could}) \cdot d(1 \mid 1, 6)$$

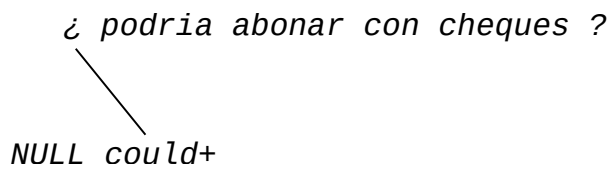


Figura 6.3: Efecto de la operación $add(\textit{could}, 1)$. El símbolo + a la derecha de *could* indica que la hipótesis está abierta.

Cabe resaltar los siguientes aspectos:

- La contribución del modelo de lenguaje pasa por considerar a *could* como primera palabra de la frase.
- La distorsión en el modelo 3 depende también de la longitud I de la frase destino. Como ya comentamos cuando presentamos la fórmula para el modelo dicha longitud se desconoce, por lo tanto suponemos que el modelo se habrá estimado sin tener en cuenta esa dependencia.
- El factor 2 es el factor combinacional que se aplica a la fertilidad, al quedar la hipótesis abierta se supone que es 2.
- El sumatorio recoge precisamente la probabilidad de que la hipótesis tenga fertilidad de al menos 2.

Contribución del operador $extend(2)$

En la figura 6.4 se muestra el efecto de la operación $extend(2)$, sobre la hipótesis. En cuanto a los factores añadidos a la *puntuación*, puede destacarse

la revisión de la fertilidad para la palabra que se está extendiendo mediante un cociente de sumatorios:

$$3\left(\frac{\sum_{\phi=3..\phi_{max}} n(\phi \mid \textit{could})}{\sum_{\phi=2..\phi_{max}} n(\phi \mid \textit{could})}\right) \cdot t(\textit{podría} \mid \textit{could}) \cdot d(2 \mid 1,6)$$

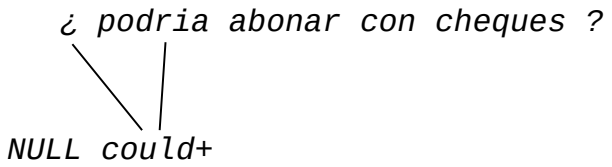


Figura 6.4: Efecto de la operación $extend(2)$.

En este caso no hay contribución del modelo de lenguaje, puesto que no se añade ninguna palabra a la hipótesis.

Contribución del operador $close()$

En la figura 6.5 aparece el efecto del operador $close()$ sobre la hipótesis, al cerrar la hipótesis se conoce la fertilidad de la última palabra destino introducida, por lo que la *puntuación* deberá actualizarse convenientemente, es decir el sumatorio y el último factor combinacional deben dividirse. Tampoco habrá en este caso contribución del modelo de lenguaje. Los términos añadidos a la *puntuación* son:

$$\frac{1}{3\sum_{\phi=3..\phi_{max}} n(\phi \mid \textit{could})} \cdot n(2 \mid \textit{could})$$

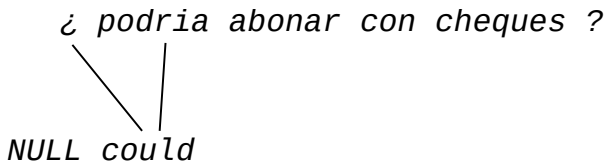


Figura 6.5: Efecto de la operación $close()$.

Contribución del operador $addZFert(I, pay, 3)$

En la figura 6.6 se muestra cómo afecta la operación $addZFert(I, pay, 3)$ a la hipótesis anterior. En cuanto a la *puntuación* de la hipótesis, se le añaden los

factores:

$$p(I \mid \$, could) \cdot p(pay \mid could, I) \times$$
$$n(0 \mid I) \cdot 2 \left(\sum_{\phi=2..\phi_{max}} n(\phi \mid pay) \right) \cdot t(abonar \mid pay) \cdot d(3 \mid 3, 6)$$

La palabra de fertilidad cero introducida añade un término a la contribución del modelo de lenguaje y un término de fertilidad al modelo de traducción. Con respecto a la palabra de fertilidad distinta de cero, los términos que se añaden se construyen de manera idéntica a la operación $add(\cdot)$.

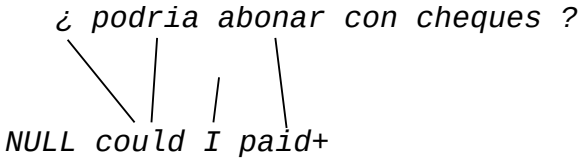


Figura 6.6: Efecto de la operación $addZFert(I, pay, 3)$.

Contribución del operador $addNull()$

El resto de la hipótesis puede construirse aplicando las reglas que se han definido, sin embargo, entre los operadores aplicados no figura $addNull(\cdot)$. No obstante, y a titulo de ejemplo, podríamos haber considerado que la palabra origen ¿ (en nuestro caso alineada con *could*), se alinease con e_o; ¿ sería, por tanto, una palabra espuria. En tal caso, la primera operación a aplicar hubiera sido $addNull(\cdot)$. La figura 6.7 muestra el alineamiento introducido por dicha operación. Cabe destacar que cada vez que se introduce una palabra espuria, se modifica la asunción sobre el valor de ϕ_0, razón por la cual no sólo multiplicamos por p_1 sino que dividimos por p_0^2. De esta manera las contribuciones a la *puntuación* de esta operación serían:

$$\binom{6-1}{1} \cdot \frac{p_1}{p_0^2} \cdot t(¿ \mid NULL)$$

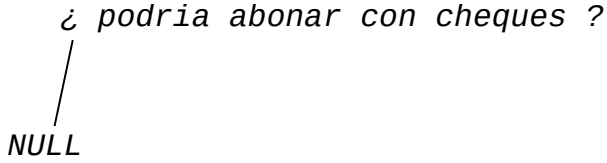


Figura 6.7: Efecto de la operación $addNull()$.

Operadores para el modelo 4.

Los operadores definidos para el modelo 3 son válidos para el modelo 4, sin embargo, el cálculo de las probabilidades sí que se ve afectado por el cambio de modelo.

El modelo 4 introduce, como ya se describió en el capítulo 3, profundos cambios en el cálculo de las distorsiones; además, las extensiones de una hipótesis abierta sólo pueden producirse con valores crecientes de j, por esta razón, los factores combinacionales que se multiplican en el modelo 3 para los operadores $add(\cdot)$, $extend(\cdot)$ y $addZFert(\cdot)$ ya no aparecen para el modelo 4.

A continuación se muestran las contribuciones de todos los operadores para el mismo ejemplo que en el apartado anterior (se omiten los gráficos con los alineamientos y las correspondientes contribuciones del modelo de lenguaje ya que no sufren ninguna variación):

$add(\textbf{\textit{could}}, 1)$**:**

$$\left(\sum_{\phi=2..\phi_{max}} n(\phi \mid could)\right) \cdot t(¿ \mid could) \cdot d_{=1}(1-0 \mid \mathcal{E}_c(NULL), \mathcal{F}_c(¿))$$

Recordemos que en

$$d_{=1}(\pi_{i1} - c_{\rho i} \mid \mathcal{E}_c(e_{\rho i}), \mathcal{F}_c(\tau_{i1}))$$

π_{i1} es el índice de la primera palabra alineada con e_i. En este caso, tenemos $i = 1$, luego $\pi_{11} = 1$. En cuanto a ρ_i, se define como la primera palabra con fertilidad mayor que cero a la izquierda de e_i. En este caso dado que $i = 1$, ρ_1 queda indefinido, con lo que en este caso particular, c_ρ se fija a cero.

$extend(2)$**:**

$$\left(\frac{\sum_{\phi=3..\phi_{max}} n(\phi \mid could)}{\sum_{\phi=2..\phi_{max}} n(\phi \mid could)}\right) \cdot t(podría \mid could) \cdot d_{>1}(2-1 \mid \mathcal{F}_c(podría))$$

En este caso, se tenía:

$$d_{>1}(\pi_{ik} - \pi_{i(k-1)} \mid \mathcal{F}_c(\tau_{ik}))$$

π_{ik} corresponde con el índice de la k-ésima palabra origen alineada con e_i, en este caso, $i = 1$ (que es el índice de la palabra de $\mathbf{e}$ que se está alineando) y $k = 2$ (que es la fertilidad de e_i). π_{ik-1} es el índice de la palabra alineada con e_i inmediatamente anterior a π_{ik}.

$close()$**:**

$$\frac{1}{\sum_{\phi=3..\phi_{max}} n(\phi \mid could)} \cdot n(2 \mid could)$$

A diferencia del anterior, en este caso solo se omite el factor combinacional.

$addZFert(\textit{I}, \textit{pay}, 3)$:

$$n(0 \mid I) \left(\sum_{\phi=2..\phi_{max}} n(\phi \mid \textit{pay}) \right) \cdot t(\textit{abonar} \mid \textit{pay}) \cdot d_1(3-2 \mid \mathcal{E}_c(\textit{could}), \mathcal{F}_c(\textit{abonar}))$$

En este caso, $i = 3$ y $\rho_3 = 1$ (porque no se tiene en cuenta la palabra de fertilidad cero). c_1 es la media de la suma de las posiciones de $\mathbf{f}$ alineadas con e_1, es decir, con *could*, aquí las posiciones alineadas son 1 y 2, siendo su centro igual a 2 (1.5 redondeado hacia arriba).

$addNull(1)$:

$$\binom{6-1}{1} \cdot \frac{p_1}{p_0^2} \cdot t(¿ \mid \textit{NULL})$$

Este operador queda idéntico al del modelo 3.

6.2.3 El proceso de expansión de hipótesis.

Una vez definidos los operadores, podemos establecer un algoritmo para expandir las hipótesis. Dicho algoritmo está fuertemente inspirado en el que se describe en [Berger et al. 96a] para el modelo 3. Como veremos, adaptar el algoritmo para ser usado con el modelo 4 es bastante sencillo.

Expansión para el modelo 3.

El proceso de expansión de hipótesis del algoritmo *stack-decoding* recorre cada una de las posiciones de la frase origen que todavía no se han alineado (al conjunto de posiciones cubiertas para la hipótesis h lo denotamos con $\mathcal{C}(h)$), y les aplica los operadores antes definidos en función de que estos satisfagan o no las precondiciones dadas en la figura 6.1. En el algoritmo 6.2 se expone el proceso de expansión en detalle.

Expansión para el modelo 4.

En lo que respecta al modelo 4, el algoritmo de expansión es idéntico, salvo en el tratamiento de hipótesis abiertas, en el que se debe verificar que las extensiones se hacen con índices crecientes de las palabras de la frase origen. Esta afirmación se justifica a la vista de la fórmula del modelo 4 (ver capítulo 3, sección 3.3.1, página 64), donde existe un parámetro de distorsión $d_{>1}$ para las palabras e_i con fertilidad mayor que 1. En esos casos se tendrán en cuenta π_{ik} y $\pi_{i(k-1)}$, que serán, respectivamente, la k-ésima palabra origen alineada con e_i, y la $(k-1)$-ésima palabra origen alineada con e_i. En el algoritmo 6.3 se expone el proceso de expansión para el modelo 4, donde con el símbolo "$\Leftarrow$" hemos marcado la parte que difiere del algoritmo de expansión para el modelo 3.

Algoritmo: ***ExpansiónM3***(hip)
entrada: $Pila : P, f_1^J$
salida: P
  para_todo $j \notin \mathcal{C}(hip)$ hacer
    si hip.es_abierta() entonces
      $hip' = hip$
      $hip'.extend(j)$; $apilar(P, hip')$ {op. extend}
      $hip'.close()$; $apilar(P, hip')$ {op. extend + close}
    sino
      para_todo $e \in \mathcal{E}$ hacer
        $hip' = hip$
        $hip'.add(e, j)$; $apilar(P, hip')$ {op. add}
        $hip'.close()$; $apilar(P, hip')$ {op. add + close}
        para_todo $ze \in \mathcal{E}$ hacer
          $hip' = hip$
          $hip'.addZFert(ze, e, j)$; $apilar(P, hip')$ {op. addZFert}
          $hip'.close()$; $apilar(P, hip')$ {op. addZFert + close}
        fin_para
      fin_para
      si $hip.\phi_0 < J/2$ entonces {Conectar j con *NULL*}
        $hip' = hip$
        $hip'.addNull(j)$; $apilar(P, hip')$ {op. addNull}
      fin_si
    fin_si
  fin_para
  devolver: P

Algoritmo 6.2: Algoritmo de expansión para el modelo 3.

Algoritmo: ***ExpansiónM4(hip)***
entrada: $Pila : P, f_1^J$
salida: P
  para_todo $j \notin \mathcal{C}(hip)$ hacer
    si hip.es_abierta() entonces
      si $j > hip.obtener\_indice\_de\_la\_palabra\_origen\_anterior()$ entonces {$\Leftarrow$}
        $hip' = hip$
        $hip'.extend(j)$; $apilar(P, hip')$ {op. extend}
        $hip'.close()$; $apilar(P, hip')$ {op. extend + close}
      fin_si
    sino
      para_todo $e \in \mathcal{E}$ hacer
        $hip' = hip$
        $hip'.add(e, j)$; $apilar(P, hip')$ {op. add}
        $hip'.close()$; $apilar(P, hip')$ {op. add + close}
        para_todo $ze \in \mathcal{E}$ hacer
          $hip' = hip$
          $hip'.addZFert(ze, e, j)$; $apilar(P, hip')$ {op. addZFert}
          $hip'.close()$; $apilar(P, hip')$ {op. addZFert + close}
        fin_para
      fin_para
      si $hip.\phi_0 < J/2$ entonces {Conectar j con *NULL*}
        $hip' = hip$
        $hip'.addNull(j)$; $apilar(P, hip')$ {op. addNull}
      fin_si
    fin_si
  fin_para
  devolver: P

Algoritmo 6.3: Algoritmo de expansión para el modelo 4.

6.2.4 Estructura de una hipótesis.

La estructura de una hipótesis viene dada por toda aquella información que se requiera para:

1. Obtener directa o indirectamente la información relativa a la solución del problema que representa la hipótesis.
2. Calcular su *puntuación* parcial asociada según lo establecido en las secciones anteriores.

Lo dicho en 1 es obvio, ya que toda hipótesis representa una solución a un determinado problema; dicha solución, en el caso que nos ocupa, consistirá en un vector compuesto por las palabras que traducen la frase de entrada. Como vamos a trabajar en todo momento con hipótesis parciales, tendremos vectores con traducciones asimismo parciales, que se irán completando a medida que se apliquen los distintos operadores sobre la hipótesis.

Ahora bien, como se comenta en la sección 6.2, se necesita un mecanismo de asignación de puntuaciones que nos diga cuan buena es una determinada hipótesis. Dicho mecanismo necesitará normalmente más información que un mero vector de palabras para funcionar adecuadamente, y estará basado en el modelo de traducción elegido.

Recordemos que para los modelos estadísticos, la *puntuación* de una hipótesis viene dado por el producto de dos contribuciones diferentes: la del modelo del lenguaje, y la del modelo de traducción. Por tanto, será necesaria cierta cantidad de información adicional para el cálculo de las puntuaciones tal y como se decía en 2, y dicha información dependerá de los modelos de lenguaje y de traducción que se utilicen.

Datos requeridos por el modelo de lenguaje

Para calcular la contribución a la *puntuación* de una hipótesis debido a un modelo de n-gramas, se requiere almacenar una historia consistente en las $n-1$ últimas palabras añadidas al vector solución.

No obstante, como dicha información se halla contenida en el vector destino, no son necesarias estructuras de datos adicionales para el modelo de lenguaje.

Datos requeridos por el modelo 3

Se requiere la siguiente información para calcular la *puntuación* asociada a una hipótesis según el modelo 3:

- Una variable donde almacenar la *puntuación* parcial asociada a la hipótesis.
- Un vector de alineamiento a_1^J, de J números naturales para reflejar cómo se van alineando las palabras de la frase origen con las de la frase destino, siendo J el número de palabras o longitud de la frase origen.

- Un indicador para indicar si la hipótesis está abierta o no.
- Un indicador para indicar si la hipótesis es completa o no. Una hipótesis es completa cuando no es abierta y además todas las palabras de la frase origen han sido alineadas.

Datos requeridos por el modelo 4

Las estructuras de datos requeridas por las hipótesis del modelo 4 son idénticas en todo a las del modelo 3 salvo por un detalle relacionado con la forma en que se modelan las extensiones en el modelo 4: si tenemos una hipótesis abierta, y la última palabra de la frase destino e_i fue alineada con la palabra origen f_j, entonces, posteriores extensiones de e_i sólo podrán hacerse con palabras $f_{j'}$ tales que $j' > j$. Esta restricción obliga a que se memorice en la hipótesis el valor de j, siendo esa la única variante introducida por el modelo 4 en la estructura de datos de las hipótesis.

6.3 Taxonomía de los algoritmos de pila

En la la sección 6.2, donde se presentó el algoritmo básico de pila, se hicieron dos importantes asunciones con respecto a su funcionamiento:

1. Que las hipótesis se almacenaban en una única cola de prioridades.
2. Que en cada iteración se escogía una y sólo una hipótesis para ser expandida. Esa hipótesis era, en concreto, aquella con mayor *puntuación* con respecto al modelo de traducción y de lenguaje escogidos.

El algoritmo básico *StackDecoding*, también llamado *algoritmo* $A^\star$ [Jelinek 69], tal como ha sido definido presenta una característica poco atractiva, consistente en que, en la mayoría de los casos, una hipótesis con menor número de palabras alineadas tendrá mayor prioridad sobre otra con más posiciones cubiertas a la hora de realizar la expansión, incluso aunque esta última ofrezca una traducción parcial mejor que la de la primera. La razón de esto es simple: cada nuevo operador aplicado sobre una hipótesis introduce invariablemente productos por números entre 0.0 y 1.0 que decrementarán significativamente la *puntuación* de la hipótesis, de manera que la *puntuación* de una hipótesis puede ser mejor que el de otra debido únicamente a que se han aplicado menor número de operadores sobre ella.

Una posible solución a este problema pasa por clasificar las hipótesis en múltiples colas, de forma que cada cola se relaciona con cada uno de los posibles subconjuntos de posiciones de la frase origen que se puedan formar (habrá por tanto un máximo de 2^J colas). En tal caso, el algoritmo de selección de hipótesis a expandir podría ser distinto al del *StackDecoding* básico, por ejemplo, podrían escogerse, según se comenta en [Germann et al. 01], las N mejores hipótesis

de cada cola. De esta manera dos hipótesis sólo podrían competir para una eventual expansión en el caso de que hubieran alineado el mismo subconjunto de posiciones de la frase origen, es decir, que pertenezcan a la misma cola.

En [Berger et al. 96a], uno de los documentos principales sobre la traducción automática estadística usando algoritmos de pila se propone un sofisticado algoritmo de múltiples pilas en el que cada una tiene un *umbral* o *threshold* asociado que establece qué hipótesis se expanden. El *umbral* se utiliza del siguiente modo: en cada iteración se recorren todas la pilas del algoritmo, que son creadas bajo demanda, seleccionándose de cada una, aquellas hipótesis cuya *puntuación* supere el valor del *umbral*. Para establecer el *umbral* se detalla un heurístico bastante complejo, que asocia un valor llamado *normalizador* a cada cola. El normalizador de una cola se define como el producto de la probabilidad de aparición para cada una de las palabras origen, de las posiciones del subconjunto asociado a la cola. Estas probabilidades se calculan como unigramas a partir de grandes textos de entrada.

Los normalizadores así definidos, se usan para calcular el *umbral* de las colas, distinguiendo entre colas *en la frontera* y colas *padre*. Se dice que una cola Q está en la frontera si no existe alguna cola, con al menos una hipótesis, cuyo conjunto de palabras alineadas de la frase origen contenga al conjunto de palabras alineadas de Q (a este conjunto le llamamos F). Una cola es una cola padre si no es una cola en la frontera. En la figura 6.8 podemos ver un ejemplo en el que aparecen colas padre y colas frontera para una frase de entrada de longitud 3, donde $F = \{[011], [101], [110]\}$.

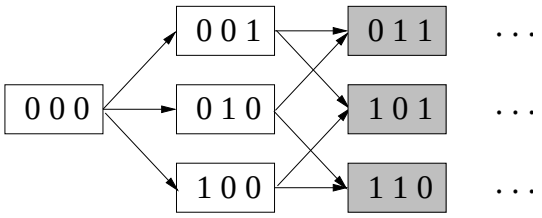


Figura 6.8: Un ejemplo de la creación de colas en un algoritmo multipila con *umbral*. Las colas frontera son las sombreadas, el resto son colas padre.

Para las colas en la frontera, según [Berger et al. 96a], el *umbral* se calcula de la siguiente manera:

$$puntuacion\_normalizada(Q) = \max_{h \in Q} \left\{ \frac{puntuacion(h)}{normalizador(Q)} \right\}$$

$$Z = \max_{Q \in F} \{puntuacion\_normalizada(Q)\}$$

$$umbral(Q) = Z \cdot puntuacion\_normalizada(Q)/cte \ ,$$

donde: Q representa una cola, y h representa una hipótesis.

Toda cola padre, habrá sido previamente cola frontera con lo que tendrá asociado un *umbral* previo. Este *umbral* se actualiza cada vez que se extiende una hipótesis h de esa cola realizando un cálculo consistente en la *puntuación* de h menos aquellas contribuciones de h correspondientes a palabras de $\mathbf{f}$ que no estén alineadas en la cola padre, dividido por una constante. Si el valor resultante de la diferencia es menor que el *umbral* previo de la cola padre, entonces es sustituido por él.

Otra aproximación multipila con *umbral* es propuesta en [Al-Onaizan et al. 99] aunque en este caso no se detalla la forma de establecer el *umbral*.

No obstante en la bibliografía pueden encontrarse otras aproximaciones, como en [Germann et al. 01] donde se sugiere la posibilidad, bien de extraer la mejor hipótesis de cada pila, o bien un número no determinado de las mejores hipótesis a nivel absoluto en toda las estructura de múltiples pilas. Incluso, en [Och et al. 01] se hace un estudio completo sobre un algoritmo de una sola pila. A este respecto cabe decir que la deficiencia expresada anteriormente sobre las limitaciones de los algoritmos de una sola pila, puede ser superada introduciendo el uso de heurísticos, que serán tratados en el apartado 6.4.2.

A modo de resumen, se da la clasificación de la figura 6.9, donde figuran las tres aproximaciones distintas de los algoritmos basados en pila.

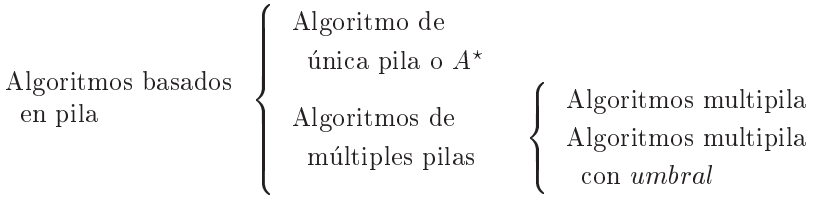


Figura 6.9: Taxonomía de los algoritmos basados en pila.

6.4 Optimizaciones en el proceso de búsqueda.

Cómo veremos en la sección siguiente el coste computacional de los algoritmos de pila en general es bastante elevado. Por ello parece razonable realizar ciertas optimizaciones sobre distintas facetas del algoritmo inicialmente propuesto en las secciones anteriores.

En los apartados siguientes se incluye tan sólo la descripción de dichas optimizaciones, posponiendo las pruebas sobre el impacto real que producen en la eficiencia del algoritmo a la sección 6.6 (discusión de los algoritmos de pila), en el que se realiza un estudio empírico de eficiencia y calidad utilizando para ello

la tarea EuTrans-I. Del mismo modo se comentan los errores de búsqueda que podrían darse lugar con cada optimización propuesta.

Al igual que comentamos en el capítulo anterior, cabe matizar que hablaremos de optimizaciones en el sentido de mejorar la eficiencia de los algoritmos, aunque ello pueda, en la mayoría de los casos, ocasionar que el algoritmo no pueda garantizar la optimalidad de la solución.

6.4.1 Reducción de la complejidad del algoritmo

Limitar el número de traducciones de una palabra f

En este caso, al igual que hacíamos con el algoritmo *DPSearch* (ver sección 5.7), no resulta práctico considerar como posibles traducciones de una palabra f todas las palabras de $\mathcal{E}$. Por tanto limitaremos ese número a las W mejores traducciones inversas, y al conjunto de ellas le llamaremos $\mathcal{W}(f)$.

Esta optimización es fundamental para mantener bajo control la complejidad del algoritmo. Si $\mathbf{e}$ es la traducción óptima de la frase $\mathbf{f}$, y en ella, la palabra destino e_i se alinea con la palabra origen f_j, entonces será preciso que e_i se encuentre en el conjunto $\mathcal{W}(f_j)$, de lo contrario, se puede garantizar que el alineamiento de la frase óptima no podrá ser alcanzado. En definitiva, esta optimización puede dar lugar a errores de búsqueda y la solución a este problema pasa por aumentar el valor de W.

Limitar el número de palabras de fertilidad cero

El bucle de operaciones *addZFert()* es con diferencia la parte más costosa del algoritmo de expansión. Al igual que en el caso anterior, tampoco es práctico en este caso considerar todas las palabras del vocabulario $\mathcal{E}$ como palabras de fertilidad cero. En la práctica lo que se hace es coger solo aquellas palabras $e' \in \mathcal{E}\ t.q.\ n(0 \mid e') > \tau$ donde τ es un cierto umbral dado como parámetro, o simplemente fijar un número máximo de palabras, al que llamaremos Z, y al conjunto de esas palabras le llamaremos $\mathcal{Z}$. Juntando esta optimización y la anterior, llegamos a una nueva forma del algoritmo de expansión, descrito en el algoritmo 6.4, mucho más cercana a la propuesta en [Berger et al. 96a].

Un efecto similar al caso anterior ocurre con las palabras de fertilidad cero y con el parámetro Z. De la misma forma, encontramos alineamientos inalcanzables con el algoritmo en caso de que falte la palabra con fertilidad cero adecuada. Es decir, esta optimización también podrá provocar errores de búsqueda. La solución a este problema pasa por aumentar el valor de Z.

Limitación del tamaño de las pilas

Tanto en un algoritmo $A^\star$ como en un algoritmo de múltiples pilas, puede establecerse un tamaño máximo de pila S, de manera que una vez alcanzado

Algoritmo: ***ExpansiónM3_opt_W+Z(hip)***
entrada: $Pila : P, f_1^J$
salida: P
  para_todo $j \notin \mathcal{C}(hip)$ hacer
    si hip.es_abierta() entonces
      $hip' = hip$
      $hip'.extend(j)$; $apilar(P, hip')$ {op. extend}
      $hip'.close()$; $apilar(P, hip')$ {op. extend + close}
    sino
      para_todo $e \in \mathcal{W}(f_j)$ hacer {$\Leftarrow$}
        $hip' = hip$
        $hip'.add(e, j)$; $apilar(P, hip')$ {op. add}
        $hip'.close()$; $apilar(P, hip')$ {op. add + close}
        para_todo $ze \in \mathcal{Z}$ hacer {$\Leftarrow$}
          $hip' = hip$
          $hip'.addZFert(ze, e, j)$; $apilar(P, hip')$ {op. addZFert}
          $hip'.close()$; $apilar(P, hip')$ {op. addZFert + close}
        fin_para
      fin_para
      si $hip.\phi_0 < J/2$ entonces {Conectar j con *NULL*}
        $hip' = hip$
        $hip'.addNull(j)$; $apilar(P, hip')$ {op. addNull}
      fin_si
    fin_si
  fin_para
  devolver: P

Algoritmo 6.4: Algoritmo de expansión para el modelo 3 con las optimizaciones W y Z. Las diferencias con el algoritmo básico se han marcado con el símbolo $\Leftarrow$.

dicho tamaño, no se inserte una hipótesis si su *puntuación* es menor que el menor de las *puntuaciones* de las hipótesis de la/s cola/s.

Si no se cumpliese esta condición, entonces la inserción habría de hacerse, lo que obligaría a podar la última hipótesis de la cola. A este respecto, es importante resaltar que existe una diferencia cualitativa entre el algoritmo de una sola pila y los de múltiples pilas en cuanto a cómo se realiza dicha poda: si usamos una sola pila, las hipótesis almacenadas en la ella compiten por no ser podadas sin tener en cuenta el número de palabras origen que hayan sido alineadas, esto tiene el mismo inconveniente que ya se mencionó en el apartado 6.3 relativo a la selección de hipótesis para la expansión, es decir, normalmente se podarán aquellas hipótesis con mayor número de palabras origen cubiertas. En los algoritmos de múltiples pilas, en cambio, esta situación no se dará debido a que

cada pila almacena precisamente aquellas hipótesis que hayan alineado el mismo subconjunto de palabras origen.

El principal interés de esta optimización consiste en mantener bajo control el coste espacial del algoritmo.

Esta optimización también puede provocar errores de búsqueda, debido a que en caso de que una pila hubiese alcanzado ese máximo de hipótesis almacenadas, la inserción de una hipótesis con mejor *puntuación* que, al menos, la última hipótesis de la pila, provocaría la eliminación de ésta. Si se produce la anterior situación y resulta que la hipótesis descartada era la hipótesis destinada (mediante la aplicación de los operadores necesarios) a convertirse en la hipótesis óptima, entonces se produce un error de búsqueda. Nuevamente, la solución a este problema pasa por aumentar del valor del parámetro.

Reducción de la complejidad de $addZFert(\cdot)$

El bucle donde se ejecuta el operador $addZFert(\cdot)$ es como ya se ha comentado, la parte más costosa del algoritmo de expansión. En [Germann et al. 01] se propone una poda basada en las contribuciones a la *puntuación* del operador $addZFert(\cdot)$ en relación con las del operador $add(\cdot)$. Veámoslo con un ejemplo:

Sea w la nueva palabra a añadir con la operación $add(w, j)$ sobre la hipótesis que se está expandiendo (e_1^i), y zw_k cada una de las posibles palabras de fertilidad cero. De este modo según el algoritmo 6.4, al aplicar el operador $add(w, j)$ e iterativamente el operador $addZFert(zw_k, w, j)$ se generará el siguiente conjunto de hipótesis:

$$
\begin{array}{l}
e_1\, e_2\, \ldots\, e_i\, w\text{+} \\
e_1\, e_2\, \ldots\, e_i\, w \\
e_1\, e_2\, \ldots\, e_i\, zw_1\, w\text{+} \\
e_1\, e_2\, \ldots\, e_i\, zw_1\, w \\
\vdots \\
e_1\, e_2\, \ldots\, e_i\, zw_Z\, w\text{+} \\
e_1\, e_2\, \ldots\, e_i\, zw_Z\, w
\end{array}
$$

La optimización consiste en aplicar la operación $addZFert(zw_k, w, j)$ siempre y cuando mejore la *puntuación* de la hipótesis parcial con respecto al operador $add(w, j)$. La contribución de ambas operaciones a la *puntuación* de la hipótesis, utilizando el modelo 4, será:
$add(w, j)$:

$$
\boldsymbol{p(w \mid e_{i-1}, e_i)} \times
$$

$$
2 \left(\sum_{\phi=2..\phi_{max}} n(\phi \mid w) \right) \cdot t(f_j \mid w) \cdot d_{=1}(j - c_{\rho_i} \mid \mathcal{E}_c(e_{\rho_i}), \mathcal{F}_c(f_j))
$$

$addZFert(zw_k, w, j)$:

$$\boldsymbol{p(zw_k \mid e_{i-1}, e_i) \cdot p(w \mid e_i, zw_k)} \times$$
$$\boldsymbol{n(0 \mid zw_k)} \cdot 2 \left(\sum_{\phi=2..\phi_{max}} n(\phi \mid w) \right) \cdot t(f_j \mid w) \cdot d_{=1}(j - c_{\rho_i} \mid \mathcal{E}_c(e_{\rho_i}), \mathcal{F}_c(f_j))$$

En negrita hemos marcado las partes que diferencian ambas contribuciones, con lo que la optimización propuesta consiste en aplicar la operación $addZFert(zw_k, w, j)$, si:

$$p(w \mid e_{i-1}, e_i) < p(zw_k \mid e_{i-1}, e_i) \cdot p(w \mid e_i, zw_k) \cdot n(0 \mid zw_k) \ .$$

Cabe matizar que:

- Esta optimización es válida únicamente si se utiliza el modelo 4 como modelo de traducción, puesto que se cumple que ρ_i es idéntico en ambos operadores, ya que tras aplicar el operador $addZFert(\cdot)$, se garantiza que e_{i-1} tiene fertilidad cero, por lo que no es tenido en cuenta para el cálculo de ρ_i. Además, precisamente por la manera en que se define ρ_i (la posición de la primera palabra de fertilidad mayor que 1 a la izquierda de e_i) vemos que las operaciones $addZFert(\cdot)$ así realizadas no producen ningún cambio en las subsiguientes distorsiones. En el algoritmo 6.5 podemos ver como queda la expansión para el modelo 4 usando la simplificación. Una vez más hemos marcado con ($\Leftarrow$) las diferencias, con respecto al algoritmo 6.4 en este caso.

- Esta optimización se convierte en un heurístico si se usan trigramas para el modelo de lenguaje, o en general, si se usan n-gramas con $n \geq 3$, como será detallado a continuación.

- Para el modelo 3, la simplificación sólo puede utilizarse como un heurístico, debido a que todas las distorsiones siguientes a la operación $addZFert(\cdot)$ resultan afectadas, es decir la contribución de ambas operaciones a la *puntuación* de la hipótesis en este caso será:

  $add(w, j)$:

  $$\boldsymbol{p(w \mid e_{i-1}, e_i)} \times$$
  $$2 \left(\sum_{\phi=2..\phi_{max}} n(\phi \mid w) \right) \cdot t(f_j \mid w) \cdot \boldsymbol{d(j \mid i+1, J)}$$

  $addZFert(zw_k, w, j)$:

  $$\boldsymbol{p(zw_k \mid e_{i-1}, e_i) \cdot p(w \mid e_i, zw_k)} \times$$
  $$\boldsymbol{n(0 \mid zw_k)} \cdot 2 \left(\sum_{\phi=2..\phi_{max}} n(\phi \mid w) \right) \cdot t(f_j \mid w) \cdot \boldsymbol{d(j \mid i+2, J)}$$

En definitiva, la optimización de la operación $addZFert(\cdot)$ propuesta para el modelo 4, se convierte en un mero heurístico para el modelo 3, a no ser que se use con un modelo de lenguaje de bigramas. En el ejemplo de la figura 6.11 se muestra el resultado de usar esta optimización tomando en consideración un modelo de lenguaje de trigramas y el modelo 4 como modelo de traducción. Sin embargo, la hipótesis óptima, ver la figura 6.10, podía ser alcanzada con el mismo traductor sin usar la optimización. Veamos con un poco más de detalle que está ocurriendo.

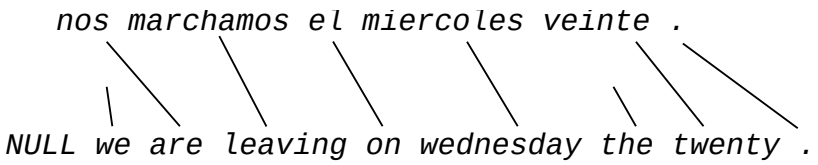


Figura 6.10: Traducción óptima con $addZFert(\cdot)$ inicial.

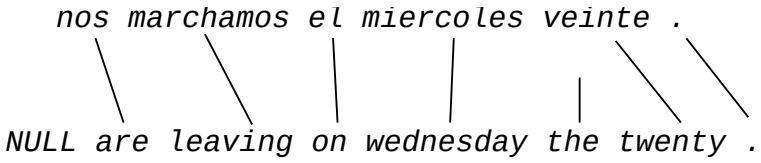


Figura 6.11: Traducción errónea debido a optimización de $addZFert(\cdot)$.

El problema con la optimización reside, como podemos imaginar, en que se decide erróneamente no efectuar una operación $addZFert(\cdot)$ (concretamente $addZFert(\textit{we}, \textit{are}, 1)$). Habrá ocasiones en que $addZFert(\cdot)$ no aporte una mejora con respecto a la operación $add(\cdot)$ en los términos que se establecieron en el apartado 6.4 con lo que será descartado, sin embargo, en cualquier operación subsiguiente que añada una palabra destino se calculará una nueva contribución del modelo de lenguaje. Si se usan trigramas como modelo de lenguaje, la historia de la frase tendrá en cuenta las dos últimas palabras añadidas, digamos u y v como la penúltima y última palabra introducida respectivamente, sin embargo, u podría haber sido la palabra de fertilidad cero descartada por la optimización y producir una drástica mejora en la contribución del modelo de lenguaje, como de hecho ocurre en el caso de la figura. No obstante, dicha operación nunca se llegó a efectuar. Lo que ocurre en la figura 6.11 se puede expresar formalmente:

$$p(\textit{we} \mid \$, \$) \cdot (\textit{are} \mid \$, \textit{we}) \cdot n(0 \mid \textit{we}) < p(\textit{are} \mid \$, \$)$$

$$p(\textit{leaving} \mid \textit{we}, \textit{are}) \gg p(\textit{leaving} \mid \$, \textit{are})$$

Algoritmo: ***ExpansiónM4_opt_W+Z+ZE****(hip)*
entrada: $Pila : P, f_1^J$
salida: P
  para_todo $j \notin \mathcal{C}(hip)$ hacer
    si hip.es_abierta() entonces
      si $j > hip.obtener\_indice\_de\_la\_palabra\_origen\_anterior()$ entonces
        $hip' = hip$
        $hip'.extend(j)$; $apilar(P, hip')$ {op. extend}
        $hip'.close()$; $apilar(P, hip')$ {op. extend + close}
      fin_si
    sino
      para_todo $e \in \mathcal{W}(f_j)$ hacer
        $hip' = hip$
        $ModLenP = p(e \mid hip'.lm\_hist())\{\Leftarrow\}$
        $hip'.add(e, j)$; $apilar(P, hip')$ {op. add}
        $hip'.close()$; $apilar(P, hip')$ {op. add + close}
        para_todo $ze \in \mathcal{Z}$ hacer
          $hip' = hip$
          si $ModLenP < puntuacion(hip'.addZFert(ze, e, j))$ entonces $\{\Leftarrow\}$
            $hip'.addZFert(ze, e, j)$; $apilar(P, hip')$ {op. addZFert}
            $hip'.close()$; $apilar(P, hip')$ {op. addZFert + close}
          fin_si
        fin_para
      fin_para
      si $hip.\phi_0 < J/2$ entonces {Conectar j con *NULL*}
        $hip' = hip$
        $hip'.addNull(j)$; $apilar(P, hip')$ {op. addNull}
      fin_si
    fin_si
  fin_para
  devolver: P

Algoritmo 6.5: Algoritmo de expansión para el modelo 4 con las optimizaciones W, Z y $addZFert(\cdot)$.

Una forma de solucionar esto para los trigramas, consiste en modificar el algoritmo de expansión para que la verificación sobre si se debe aplicar $addZFert(\cdot)$ se posponga al momento en que se añade la siguiente palabra a la frase destino. Dicha optimización no produce ningún error de búsqueda, sin embargo, está lejos de ser tan efectiva en términos de eficiencia como la optimización inicial.

En el algoritmo 6.6 podemos ver como queda la expansión en este caso para el modelo 3. La función $se\_puede\_descartar\_addZFert()$ verifica, siguiendo con el ejemplo anterior, que:

$$p(we \mid \$,\$)\cdot p(are \mid \$, we)\cdot n(0, we)\cdot p(leaving \mid we, are) < p(are \mid \$,\$)\cdot p(leaving \mid \$, are)$$

para el caso concreto del ejemplo de la figura 6.11.

Limitación del máximo de palabras a alinear por expansión.

Una nueva simplificación, también propuesta en [Berger et al. 96a], consiste en limitar el número máximo de posiciones j aún no alineadas en cada expansión a un valor determinado, que denotaremos con A. Esto tendrá su correspondiente reflejo en el proceso de expansión. En el algoritmo 6.7, se describe cómo afectaría esta optimización al algoritmo de expansión para el modelo 3. Por claridad, se incluye esta optimización al algoritmo 6.4(*ExpansiónM3_opt_W+Z*), marcando una vez más las diferencias con respecto a él.

El parámetro A establece el número máximo de palabras de la frase origen, empezando por la primera(f_1), que se alinean en cada expansión. Esto tiene como resultado un menor número de hipótesis generadas, y también, un menor número de colas que como máximo se pueden obtener. Ésta última característica supone una reducción bastante importante del espacio de búsqueda. Supongamos que queremos traducir la frase $\mathbf{f}$ de 10 palabras y fijamos el valor de A a 4. Si partimos de la hipótesis vacía y la sometemos a nuestro algoritmo de expansión, los subconjuntos de palabras origen alineadas resultantes, que representaremos como un número binario en el que un 1 indica que la palabra está cubierta, serían:

f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0

Después de expandir la hipótesis vacía, cualquier otra hipótesis tendrá un 1 en alguna de las 4 primeras posiciones, por lo que es imposible de esta manera generar hipótesis tales que la primera palabra origen alineada no esté entre las cuatro primeras palabras de $\mathbf{f}$. En otras palabras, no pueden alcanzarse 2^6

Algoritmo: ***ExpansiónM3_opt_W+Z+ZE(hip)***
entrada: $Pila : P, f_1^J$
salida: P
  para_todo $j \notin \mathcal{C}(hip)$ hacer
    si hip.es_abierta() entonces
      $hip' = hip$
      $hip'.extend(j)$; $apilar(P, hip')$ {op. extend}
      $hip'.close()$; $apilar(P, hip')$ {op. extend + close}
    sino
      para_todo $e \in \mathcal{W}(f_j)$ hacer
        $hip' = hip$
        $hip'.add(e, j)$; $apilar(P, hip')$ {op. add}
        $hip'.close()$; $apilar(P, hip')$ {op. add + close}
        para_todo $ze \in \mathcal{Z}$ hacer
          $hip' = hip$
          si $not(hip'.ultima\_op\_fue\_addZFert())or$
            $(hip'.ultima\_op\_fue\_addZFert()and$
            $not(hip'.se\_puede\_descartar\_addZFert())$ entonces {$\Leftarrow$}
            $hip'.addZFert(ze, e, j)$; $apilar(P, hip')$ {op. addZFert}
            $hip'.close()$; $apilar(P, hip')$ {op. addZFert + close}
          fin_si
        fin_para
      fin_para
      si $hip.\phi_0 < J/2$ entonces {Conectar j con *NULL*}
        $hip' = hip$
        $hip'.addNull(j)$; $apilar(P, hip')$ {op. addNull}
      fin_si
    fin_si
  fin_para
  devolver: P

Algoritmo 6.6: Algoritmo de expansión para el modelo 3 optimizado para la operación $addZFert(\cdot)$.

Algoritmo: ***ExpansiónM3_opt_W+Z+A*** *(hip)*
entrada: $Pila : P, f_1^J$
salida: P
  para_todo $j \notin \mathcal{C}(hip)$ hacer
    si $j < j + A$ entonces $\{\Leftarrow\}$
      $A = A - 1\{\Leftarrow\}$
      si hip.es_abierta() entonces
        $hip' = hip$
        $hip'.extend(j)$; $apilar(P, hip')$ {op. extend}
        $hip'.close()$; $apilar(P, hip')$ {op. extend + close}
      sino
        para_todo $e \in \mathcal{W}(f_j)$ hacer
          $hip' = hip$
          $hip'.add(e, j)$; $apilar(P, hip')$ {op. add}
          $hip'.close()$; $apilar(P, hip')$ {op. add + close}
          para_todo $ze \in \mathcal{Z}$ hacer
            $hip' = hip$
            $hip'.addZFert(ze, e, j)$; $apilar(P, hip')$ {op. addZFert}
            $hip'.close()$; $apilar(P, hip')$ {op. addZFert + close}
          fin_para
        fin_para
        si $hip.\phi_0 < J/2$ entonces {Conectar j con *NULL*}
          $hip' = hip$
          $hip'.addNull(j)$; $apilar(P, hip')$ {op. addNull}
        fin_si
      fin_si
    fin_si
  fin_para
  devolver: P

Algoritmo 6.7: Algoritmo de expansión para el modelo 3 con la optimización A.

colas. Esto es obviamente una importante fuente de errores de búsqueda, como por ejemplo en la frase de la figura 6.12.

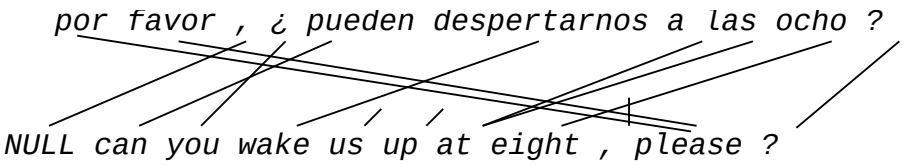


Figura 6.12: Frase problemática para un valor de A igual a 4.

Para resolver el problema que aparece en la figura, basta con subir el valor de A a 5.

Precálculo de las sumas de fertilidad

Como vimos en el apartado sobre la definición de los operadores, en aquellos casos en que una hipótesis queda abierta, se añade el factor $\sum_{\phi=k}^{\phi_{max}} n(\phi \mid e)$ a la *puntuación* (siendo k la fertilidad de la última palabra añadida más 1), debido a que todavía no se conoce la fertilidad que finalmente tendrá la última palabra introducida. La optimización aquí propuesta consiste en precalcular $m(k \mid e) = \sum_{\phi=k}^{\phi_{max}} n(\phi \mid e)$ antes de comenzar el proceso de traducción. Esta optimización, evidentemente, no afectará a los errores de búsqueda del algoritmo.

6.4.2 Reducción del espacio de búsqueda

Como todos los algoritmo basados en la técnica de ramificación y poda, los algoritmos de pila son algoritmos de búsqueda exhaustiva. A continuación veremos un par de técnicas que evitan la exploración de algunos conjuntos de hipótesis, utilizando dos tipos de poda, la recombinación de hipótesis equivalentes y la utilización de heurísticos para estimar el resto del coste de expansión de las hipótesis.

Recombinación de hipótesis

Una optimización de gran efectividad consiste en la recombinación de hipótesis. Si tenemos dos hipótesis que no pueden distinguirse a través de sus estados del modelo de lenguaje y del modelo de traducción, entonces podemos quedarnos con aquella que tenga una mayor *puntuación*. Para poder llevar a cabo la recombinación será necesario por tanto, determinar correctamente qué información de la estructura de datos de la hipótesis determina su estado, tanto para el modelo de traducción como para el modelo de lenguaje. El estado correspondiente al modelo de traducción depende de:

- El conjunto de las posiciones de la frase origen que han sido alineadas. Lo denotaremos mediante $\mathcal{C}$.
- El valor del indicador que indica si la hipótesis es completa.
- El valor del indicador que indica si la hipótesis es abierta.
- El número de palabras origen alineadas con la palabra vacía (cantidad de palabras espurias).
- La posición de la última palabra origen alineada (sólo para los modelos 4 y 5).

En cuanto al estado del modelo de lenguaje, viene dado por las $n-1$ últimas palabras añadidas a la estructura destino, para un modelo de n-gramas.

La recombinación permite reducir el número de hipótesis que se generan en un factor de 4, lo que (como veremos en los experimentos realizados) redundará en una sustancial ganancia en el tiempo de ejecución.

Uso de heurísticos en el proceso de búsqueda.

Hasta ahora hemos asumido que la función que asigna la *puntuación* a una hipótesis se basa únicamente en las palabras origen que dicha hipótesis ha alineado. A dicha función le llamaremos $Q(n)$, donde n representa a la hipótesis. No obstante, sería de interés tener una medida del coste que tiene alinear el resto de palabras. Dado que no es posible conocer exactamente dicho coste, se recurre al uso de de heurísticos, que representaremos con $h(n)$. Nuestro mecanismo de asignación de la *puntuación* quedaría entonces como $Q(n) + h(n)$.

$h(n)$ presenta la siguiente forma:

$$H^X(n) = \prod_{j \notin \mathcal{C}(n)} h^X(j) \ , \tag{6.3}$$

donde:

- X representa el identificador que se da al heurístico.
- n es la hipótesis a la que se aplica H.
- $\mathcal{C}(n)$ representa aquellas palabras de la frase origen que se han alineado.

El uso de heurísticos permite paliar el problema que se mencionaba en el apartado 6.3, relacionado con la selección de la hipótesis a expandir, dado que las hipótesis con menor número de palabras cubiertas ven penalizada su *puntuación* con la estimación del coste de alinear el resto de posiciones que hace el heurístico. De la misma manera, el problema de la poda de hipótesis para el algoritmo de una única pila también se ve reducido.

Heurísticos admisibles y heurísticos empíricos. Se dice que un heurístico es admisible [Och et al. 01] si nunca subestima el coste de completar una hipótesis cualquiera.

Un heurístico empírico [Och et al. 01] será aquel que se basa en información generada en el proceso de traducción de una frase usando un heurístico admisible. Para ello, se realiza la búsqueda normalmente; si la búsqueda fracasa (se habla aquí de fracaso como exceder un máximo de hipótesis almacenadas en la pila), entonces el algoritmo se reinicia usando la información que se recopiló en la primera pasada de traducción para calcular un heurístico.

A continuación se detallan los heurísticos que hemos utilizado.

El heurístico admisible T. La forma más sencilla de implementar un heurístico es basándose únicamente en información del modelo léxico, tal función, denotada aquí con $h^T(j)$ tendrá la siguiente forma:

$$h^T(j) = \max_e \{t(f_j \mid e)\} \tag{6.4}$$

El heurístico admisible TF. La función anterior puede ser refinada introduciendo información sobre la probabilidad de las fertilidades de las palabras destino:

$$h^{TF}(j) = \max \left\{ \max_{e \neq e_0, \phi} \left\{ t(f_j \mid e) \sqrt[\phi]{n(\phi \mid e)} \right\}, t(f \mid e_0) \right\} \tag{6.5}$$

Para la introducción de palabras espurias no se considera el parámetro de fertilidad. Cuando la fertilidad ϕ_i de una palabra destino e_i sea mayor que uno, se calcula la raíz ϕ_i-ésima, lo que evita que consideremos ϕ_i veces su fertilidad.

El heurístico admisible TFL. Por último, puede tenerse en cuenta en el cálculo del heurístico la probabilidad del modelo de lenguaje. Para ello, se considera que, para cada palabra destino e a introducir existe una probabilidad óptima $\hat{p}(e)$ del modelo del lenguaje, definida como:

$$\hat{p}(e) = \max_{u,v} \{p(e \mid u, v)\} \tag{6.6}$$

Ésta función puede incorporarse a nuestro heurístico anterior del siguiente modo:

$$h^{TFL}(j) = \max \left\{ \max_{e \neq e_0, \phi} \left\{ t(f_j \mid e) \sqrt[\phi]{n(\phi \mid e) \cdot \hat{p}(e)} \right\}, t(f \mid e_0) \right\} \tag{6.7}$$

El coste del cálculo de este heurístico hace indispensable que se precalcule antes de iniciar el proceso de traducción.

Integración del heurístico en los algoritmos de pila. El heurístico ha de añadirse a la *puntuación* de la hipótesis durante el proceso de expansión, inmediatamente después de aplicarle un operador. En el algoritmo 6.8 se muestra cómo queda la expansión para el modelo 3 usando heurísticos. En este algoritmo, hemos utilizado la función *añadirHeur(H,hip)* para añadir el valor del heurístico H a la hipótesis *hip*.

Por otro lado, habrá que sustraer el heurístico cada vez que se extrae una hipótesis de la cola para la expansión. De esta manera nos aseguramos que los heurísticos se aplican siempre sobre valores reales de la *puntuación* de una hipótesis.

La tarea de añadir y sustraer heurísticos, tal y como están definidos, es bastante sencilla, ya que la función que los calcula sólo depende del conjunto de palabras origen que todavía no han sido alineadas. Por ejemplo, en un algoritmo con múltiples pilas, siempre se añade el mismo heurístico a todas la hipótesis que se van a almacenar en una misma cola.

Fuerza predictiva de un heurístico Para evaluar la fuerza de un heurístico en el ámbito de la traducción automática estadística, se propone [Och et al. 01] comparar la *puntuación* asignada a la hipótesis vacía mediante el heurístico, con la *puntuación* de la hipótesis óptima según el modelo que se esté considerando. Una función heurística será mejor cuánto menor sea la diferencia entre ambos valores.

Errores de búsqueda relacionados con el uso de heurísticos. Los únicos errores de búsqueda debidos al uso de heurísticos aparecen al usar heurísticos empíricos, ya que estos no garantizan que nunca se subestime la *puntuación* de una hipótesis cualquiera, a diferencia de los heurísticos admisibles. Por este motivo, en este estudio solo utilizaremos heurísticos admisibles.

6.4.3 Errores de búsqueda inherentes al algoritmo.

Llamamos *error de búsqueda inherente al algoritmo* a aquellos errores de búsqueda que se producen debido a las características fundamentales del algoritmo que se usa para hacer la traducción.

Supongamos que estamos tratando de traducir al inglés la frase *despiérteme mañana a las dos* ., de la tarea EuTrans-I.

La traducción óptima de dicha frase tiene el alineamiento dado en la figura 6.13. Obsérvese que la frase destino tiene dos palabras de fertilidad cero seguidas (*me* y *up*), sin embargo, no existe ninguna operación, entre todas las que se han definido, que pueda insertar dos palabras de fertilidad cero de manera consecutiva, ni tampoco combinación alguna de operaciones que nos permita conseguirlo. Este es un ejemplo de error inherente al algoritmo, ya que resolverlo nos obliga a cambiar la forma en que trabaja. Otro tipo de error inherente al

Algoritmo: ***ExpansiónM3_opt_W+Z+H****(hip)*
entrada: $Pila : P, f_1^J$
salida: P
  para_todo $j \notin \mathcal{C}(hip)$ hacer
    si hip.es_abierta() entonces
      $hip' = hip$
      $hip'.extend(j)$; $añadirHeur(H, hip')$; $apilar(P, hip')$
      $hip'.close()$; $añadirHeur(H, hip')$; $apilar(P, hip')$
    sino
      para_todo $e \in \mathcal{W}(f_j)$ hacer
        $hip' = hip$
        $hip'.add(e, j)$; $añadirHeur(H, hip')$; $apilar(P, hip')$
        $hip'.close()$; $añadirHeur(H, hip')$; $apilar(P, hip')$
        para_todo $ze \in \mathcal{Z}$ hacer
          $hip' = hip$
          $hip'.addZFert(ze, e, j)$; $añadirHeur(H, hip')$; $apilar(P, hip')$
          $hip'.close()$; $añadirHeur(H, hip')$; $apilar(P, hip')$
        fin_para
        si $hip.\phi_0 < J/2$ entonces {Conectar j con *NULL*}
          $hip' = hip$
          $hip'.addNull(j)$; $apilar(P, hip')$
        fin_si
      fin_para
    fin_si
  fin_para
  devolver: P

Algoritmo 6.8: Algoritmo de expansión para el modelo 3 utilizando heurísticos.

algoritmo consiste cuando una frase termina con una palabra de fertilidad cero, construcción que tampoco se puede lograr con los operadores definidos.

De acuerdo a la experimentación llevada a cabo hemos observado que la proporción de errores inherentes al algoritmo es bastante grande (p.e. en torno al 20% de las frases que se usaron para testear los algoritmos contenían dos o

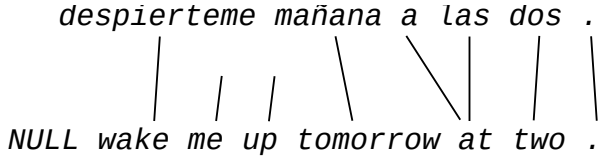


Figura 6.13: Alineamiento de una frase con dos palabras de fertilidad cero consecutivas.

más palabras de fertilidad cero consecutivas para la tarea EuTrans-I), lo cual nos lleva a proponer alguna solución a este problema.

En concreto se proponen dos soluciones diferentes:

1. Introducir la operación $reverseAddZFert(e, j)$ que viene a complementar a la operación $addZFert(ze, e, j)$, la única diferencia entre ambas es que $reverseAddZFert(\cdot)$ introduce en primer lugar la palabra de fertilidad mayor que cero e, y a continuación ze. Juntas permiten obtener el alineamiento óptimo para frases que introduzcan un máximo de dos palabras de fertilidad cero consecutivas. Además, $reverseAddZFert(\cdot)$ permite acabar una frase con una palabra de fertilidad cero.

2. Introducir la operación $add2ZFert(ze_1, ze_2, e, j)$, que añade directamente dos palabras de fertilidad cero. Esta aproximación también es útil para solucionar los errores inherentes cuando aparecen dos palabras de fertilidad cero seguidas en el alineamiento.

Como puede apreciarse, las soluciones anteriores no son generales, ya que podría darse el caso de que aparecieran en una misma frase más de dos palabras de fertilidad cero consecutivas, no obstante, la práctica nos ha demostrado que esta situación es poco frecuente, al menos para la traducción del español al inglés. Por otro lado, un intento de resolver esta situación en general pasaría por introducir una serie de operaciones $addNZFert$, aunque ello conllevaría a incrementar sustancialmente el coste del algoritmo, dado que dichas operaciones son las más costosas dentro del proceso de expansión con mucha diferencia.

En cuanto a las ventajas e inconvenientes de escoger un operador u otro, advertir que la optimización sobre $addZFert(\cdot)$ comentada en el apartado 6.4.1 no es aplicable a la operación $reverseAddZFert(\cdot)$, pero si a $add2ZFert(\cdot)$. Por otra parte la inclusión de la operación $add2ZFert(\cdot)$ nos obliga a introducir un nuevo bucle anidado en el algoritmo de expansión. En el algoritmo 6.9 podemos ver como queda la expansión si se usa $reverseAddZFert(\cdot)$ y en el algoritmo 6.10 se muestra la expansión en caso de usar $add2ZFert(\cdot)$, ambos para el modelo 3. Una vez más, y por claridad, hemos marcado con el símbolo $\Leftarrow$ las diferencias con respecto al algoritmo 6.4(*ExpansiónM3_opt_W+Z*),

En vista de los resultados experimentales es preferible utilizar la aproximación con $add2ZFert(\cdot)$, porque gracias a la simplificación son muy pocas las veces que el operador llega a aplicarse, y sin embargo se consigue alcanzar la solución óptima.

Que sepamos, los dos anteriores son los únicos tipos de errores inherentes al algoritmo que existen para los algoritmos de pila que hemos definido.

6.5 Complejidad computacional

A continuación haremos un estudio sobre la complejidad por iteración de cada uno de los algoritmos de pila, teniendo y sin tener en cuenta las optimizaciones

Algoritmo: ***ExpansiónM3_opt_W+Z+rZE****(hip)*
entrada: $Pila : P, f_1^J$
salida: P
  para_todo $j \notin \mathcal{C}(hip)$ hacer
    si hip.es_abierta() entonces
      $hip' = hip$
      $hip'.extend(j);\ apilar(P, hip')$
      $hip'.close();\ apilar(P, hip')$
    sino
      para_todo $e \in \mathcal{W}(f_j)$ hacer
        $hip' = hip$
        $hip'.add(e, j);\ apilar(P, hip')$
        $hip'.close();\ apilar(P, hip')$
        para_todo $ze \in \mathcal{Z}$ hacer
          $hip' = hip$
          $hip'.addZFert(ze, e, j);\ apilar(P, hip')$
          $hip'.close();\ apilar(P, hip')$
          $hip' = hip$
          $hip'.reverseAddZFert(ze2, ze, e, j);\ apilar(P, hip')$ $\{\Leftarrow\}$
          $hip'.close();\ apilar(P, hip')$ $\{\Leftarrow\}$
        fin_para
      fin_para
      si $hip.\phi_0 < J/2$ entonces {Conectar j con *NULL*}
        $hip' = hip$
        $hip'.addNull(j);\ apilar(P, hip')$
      fin_si
    fin_si
  fin_para
  devolver: P

Algoritmo 6.9: Algoritmo de expansión para el modelo 3 con la operación $reverseAddZFert(\cdot)$.

comentadas en la sección anterior. El tamaño del problema se da en función de la longitud en número de palabras de la frase a traducir, y se representa con la letra J.

La tabla 6.1 muestra la complejidad para los algoritmos sin optimizaciones, donde N es el número de hipótesis a expandir en el algoritmo múltiples pilas, y $f()$ es la función que calcula el *umbral* para una pila. Además, con respecto a los algoritmos con múltiples pilas cabe matizar que:

- Para el algoritmo sin *umbral* se ha supuesto que se escogen las N mejores hipótesis de todas las colas, lo que exige recorrer N veces la cima de cada una de las colas.

Algoritmo: ***ExpansiónM3_opt_W+Z+2ZE***(hip)
entrada: $Pila : P, f_1^J$
salida: P
  para_todo $j \notin \mathcal{C}(hip)$ hacer
    si hip.es_abierta() entonces
      $hip' = hip$
      $hip'.extend(j);\ apilar(P, hip')$
      $hip'.close();\ apilar(P, hip')$
    sino
      para_todo $e \in \mathcal{W}(f_j)$ hacer
        $hip' = hip$
        $hip'.add(e, j);\ apilar(P, hip')$
        $hip'.close();\ apilar(P, hip')$
        para_todo $ze \in \mathcal{Z}$ hacer
          $hip' = hip$
          $hip'.addZFert(ze, e, j);\ apilar(P, hip')$
          $hip'.close();\ apilar(P, hip')$
          para_todo $ze2 \in \mathcal{Z}$ hacer $\{\Leftarrow\}$
            $hip' = hip\{\Leftarrow\}$
            $hip'.add2ZFert(ze2, ze, e, j);\ apilar(P, hip')\ \{\Leftarrow\}$
            $hip'.close();\ apilar(P, hip')\{\Leftarrow\}$
          fin_para
        fin_para
      fin_para
      si $hip.\phi_0 < J/2$ entonces {Conectar j con *NULL*}
        $hip' = hip$
        $hip'.addNull(j);\ apilar(P, hip')$
      fin_si
    fin_si
  fin_para
  devolver: P

Algoritmo 6.10: Algoritmo de expansión para el modelo 3 con la operación $add2ZFert(\cdot)$.

- Para el algoritmo con *umbral*, se recorren todas las colas una única vez, siendo necesario ejecutar una función que nos recalcule su *umbral* asociado.

Podemos adelantar a la vista de las expresiones de la complejidad, que los algoritmos de múltiples pilas no tienen ningún interés a priori en comparación con el algoritmo de una única pila para valores de J grandes (asintóticamente hablando). El término exponencial que aparece se debe a que, para el caso peor, tendremos 2^J colas activas al inicio de una iteración. En cualquier caso, hay que decir que en la práctica, el tamaño del problema estará muy limitado, ya

que de lo contrario obtendríamos unos tiempos de traducción altísimos. Por ejemplo en [Germann et al. 01] se da un tiempo de traducción medio para frases de longitud 15 superior a 30 minutos. Esto nos hace establecer una cota superior para J por debajo de esa cifra.

Un gran problema de todos los algoritmos es el término logarítmico referido al coste de insertar hipótesis en la/s pila/s (el logaritmo se debe a la presunción de que las pilas se implementan internamente como árboles binarios de búsqueda). La x es el tamaño de la pila, valor que, de no usar optimizaciones no estará acotado. Este término nos llama especialmente la atención sobre la necesidad de limitar el tamaño de las pilas.

	Complejidad por iteración
Algoritmo de pila	$J \mid \mathcal{E} \mid^2 \cdot 2\log(x)$
Algoritmo multi pila	$N2^J + J \mid \mathcal{E} \mid^2 \cdot 2\log(x)$
Algoritmo multi pila con *umbral*	$2^J f() + J \mid \mathcal{E} \mid^2 \cdot 2\log(x)$

Tabla 6.1: Complejidad de los algoritmos de pila sin optimizaciones.

Por su parte, la tabla 6.2 muestra cómo queda la complejidad de los algoritmos cuando se introducen las optimizaciones propuestas. A continuación se recuerda el significado de los símbolos:

A : Máximo número de palabras a alinear.

W : Máximo número de traducciones inversas.

Z : Máximo número de palabras de fertilidad cero.

S : Máximo tamaño de la/s pila/s.

Cabe hacer una matización con respecto a la restricción del máximo número de palabras a alinear, ya que en ese caso se limita la cantidad de pilas que pueden generarse. Por tanto, la complejidad de los algoritmos de múltiples pilas es en realidad inferior a 2^J. Obsérvese que la complejidad es constante para el algoritmo de una única pila, sin embargo, es de gran importancia advertir que el tamaño máximo de la pila S deberá ser mucho mayor en el algoritmo de una única pila que en los de múltiples pilas, o de lo contrario estaremos expuestos a cometer un número mucho mayor de errores de búsqueda. Esto eleva el coste de cada inserción en los algoritmos de una sola pila.

Gracias al establecimiento del parámetro S la complejidad temporal y espacial del algoritmo se mantienen bajo control.

No queda clara sin embargo, la influencia del tamaño del problema en el número de iteraciones que se realizan, aunque la relación es estrecha, según se aprecia en los experimentos realizados.

	Complejidad por iteración
Algoritmo de pila	$AWZ \cdot 2\log(S)$
Algoritmo multi pila	$N2^J + AWZ \cdot 2\log(S)$
Algoritmo multi pila con *umbral*	$2^J f() + AWZ \cdot 2\log(S)$

Tabla 6.2: Complejidad de los algoritmos de pila con optimizaciones.

6.6 Discusión de los algoritmos de pila

En esta sección vamos a discutir las cualidades de los algoritmos de pila experimentalmente. Para ello llevaremos a cabo una serie de experimentos utilizando el mismo corpus de test que para el resto de algoritmos de búsqueda (ver sección 5.8, tabla 5.1,página 127), de la tarea EuTrans-I , con la única finalidad de poder realizar una comparación entre ellos objetivamente. Del mismo modo el estudio se realizará siguiendo las mismas etapas, en concreto:

1. Establecer un algoritmo sobre el que se realizarán las pruebas.

2. Hacer un recorrido por las distintas optimizaciones planteadas, experimentando sobre su efecto en la eficiencia y la tasa de aciertos del algoritmo. En concreto se harán diversas pruebas con los valores del parámetro A, el valor de N para el algoritmo de múltiples pilas sin *umbral*, el número máximo de traducciones inversas W, etcétera.

3. En tercer lugar, y usando el algoritmo inicial, haremos una comparativa sobre la influencia que tienen en la eficiencia, el algoritmo que se escoja, el modelo de traducción, el heurístico y el tamaño de frase.

Al igual que en el capítulo anterior, los modelos de traducción se han entrenado con el conjunto de entrenamiento mostrado en la tabla 2.2, es decir un corpus de 10,000 pares de frases, utilizando el esquema de entrenamiento $1^5 2^5 3^5 4^5 5^5$. Los modelos de lenguaje utilizados también han sido los mismos que los utilizados para los algoritmos basados en programación dinámica, es decir, trigramas suavizados con la técnica de descuento *Good Turing*. Del mismo modo, como resultado de este estudio, escogeremos un traductor óptimo con el que obtendremos más resultados en la sección siguiente.

En las tablas relativas a los experimentos que expondremos a continuación se incluye:

- El tiempo medio de ejecución en segundos (**secs**) por frase.

- El modelo de error, es decir: porcentaje errores de búsqueda (**ErrBusq**), porcentaje errores del modelo (**ErrModel**) y porcentaje de aciertos (**Aciertos**).

- Las tasas de error (WER y PER), en tanto por ciento de error.

6.6.1 Establecimiento de los parámetros de los algoritmos de pila

Vamos a fijar por defecto (y en base a experimentaciones preliminares) las características básicas del traductor de la siguiente manera:

- Algoritmo elegido: $A^\star$
- Modelo de traducción utilizado: modelo 4
- Parámetros: $A = 5$, $N = 10$, $S = 100000$, $W = 12$ y $Z = 24$
- Heurístico utilizado: TFL
- Recombinación: Sí
- Optimización de $addZFert$: Sí, con optimización pospuesta
- Uso de $add2ZFert$: Sí, con optimización heurística

Influencia de número máximo de palabras a alinear por expansión (parámetro A)

En la tabla 6.3 podemos ver la influencia del parámetro A con respecto a la eficiencia y aciertos en la traducción. Como era de esperar a medida que aumenta el valor A el número de errores de búsqueda disminuye y por tanto mejora la calidad de la traducción. Por contra, a medida que aumenta el valor del parámetro también aumenta el tiempo medio de traducción por frase.

A	segs	ErrBusq	ErrModel	Aciertos	WER	PER
1	0.78	80.94	11.37	7.69	26.02	22.61
2	3.23	44.15	30.43	25.42	17.33	15.11
3	8.37	32.78	35.79	31.44	16.15	14.21
4	11.69	20.74	42.81	36.45	12.47	11.29
5	17.32	19.06	42.81	38.13	12.16	11.05
6	22.72	17.73	42.81	39.46	11.50	10.80
7	26.71	17.73	42.81	39.46	11.57	10.87
8	28.48	17.73	42.81	39.46	11.57	10.87
9	28.59	17.73	42.81	39.46	11.57	10.87

Tabla 6.3: Influencia del número máximo de palabras a alinear por expansión (parámetro A).

Como podemos ver en la tabla de resultados, los resultados no mejoran a partir de $A = 6$, mientras que el tiempo requerido sí. De este modo elegir un valor de $A = 6$ parece más que suficiente. De estos resultados podemos deducir

que, al menos en las frases de test utilizadas, no ha habido alineamientos entre palabras con más de 6 posiciones de diferencia.

Influencia del número de traducciones inversas (parámetro W)

El parámetro W está, como puede apreciarse en la tabla 6.4, íntimamente ligado a la calidad de la traducción y a la eficiencia. Podemos ver que para valores superiores a $W = 30$ no mejoran los resultados y si aumenta el tiempo de ejecución. Por tanto, podemos decir que un valor de $W = 20$ sería aceptable.

W	segs	ErrBusq	ErrModel	Aciertos	WER	PER
5	11.18	25.75	39.46	34.78	13.37	12.26
10	14.36	19.06	42.81	38.13	12.19	11.08
15	20.55	18.73	43.14	38.13	12.12	11.01
20	26.72	18.06	43.14	38.80	11.84	10.73
25	29.58	17.39	43.14	39.46	11.91	10.80
30	32.79	17.06	43.14	39.80	11.84	10.73
35	36.38	17.06	43.14	39.80	11.88	10.77
40	39.53	17.06	43.14	39.80	11.88	10.77
45	43.02	17.06	43.14	39.80	11.88	10.77
50	45.38	16.39	43.81	39.80	11.88	10.77

Tabla 6.4: Influencia del número de traducciones inversas para una palabra origen dada (parámetro W).

Influencia del número de palabras de fertilidad cero (parámetro Z)

El parámetro Z establece el número máximo de traducciones de palabras de fertilidad cero que se consideran al ejecutar el bucle de operaciones $addZFert$, es por tanto un parámetro vital del algoritmo debido a su influencia en la calidad de la traducción y en la eficiencia. En la tabla 6.5 se muestran pruebas con distintos valores de este parámetro. Como puede apreciarse, este parámetro tiene un impacto todavía mayor en la eficiencia y en los aciertos que W.

A la vista de los resultados, podemos ver que para valores superiores a $Z = 30$ las mejoras son prácticamente despreciables, y si aumenta el tiempo de ejecución de forma apreciable. Por tanto, podemos decir que un valor de $Z = 30$ sería idóneo.

Z	segs	ErrBusq	ErrModel	Aciertos	WER	PER
5	3.46	55.52	26.76	17.73	18.62	17.40
10	5.66	41.14	32.44	26.42	15.80	14.69
15	9.67	35.45	35.45	29.10	14.90	13.62
20	14.77	31.44	37.79	30.77	13.72	12.50
25	18.99	18.39	43.48	38.13	12.02	10.91
30	24.07	13.71	45.48	40.80	11.64	10.56
35	28.17	13.71	45.48	40.80	11.64	10.56
40	32.79	13.71	45.48	40.80	11.64	10.56
45	40.13	13.04	45.82	41.14	11.57	10.49
50	45.04	13.04	45.82	41.14	11.57	10.49

Tabla 6.5: Influencia del número de palabras de fertilidad cero consideradas (parámetro Z).

Influencia del tamaño de las pilas (parámetro S)

Este parámetro limita el número máximo de hipótesis que puede almacenar la pila. Como podemos ver en la tabla 6.6 también afecta de forma directa a la calidad y eficiencia de la traducción.

S	segs	ErrBusq	ErrModel	Aciertos	WER	PER
1	0.33	80.94	9.03	10.03	41.96	26.64
5	1.01	60.54	22.74	16.72	32.06	24.77
10	1.30	51.17	28.09	20.74	27.06	20.04
50	3.18	29.10	37.12	33.78	17.09	14.28
100	4.60	23.75	40.13	36.12	15.11	12.75
500	8.98	20.07	42.14	37.79	12.82	11.50
700	9.53	19.06	42.81	38.13	12.19	11.05
1000	10.20	19.06	42.81	38.13	12.19	11.05
5000	12.92	19.06	42.81	38.13	12.16	11.05
10000	14.04	19.06	42.81	38.13	12.16	11.05

Tabla 6.6: Influencia del tamaño de las pilas (parámetro S).

Podemos ver que valores superiores a $S = 700$ no mejoran los resultados y si aumenta ligeramente el tiempo de ejecución, con lo que podemos decir que ese sería un valor razonable.

Influencia del número de hipótesis a expandir (parámetro N)

En la tabla 6.7 podemos ver el impacto del parámetro N en la eficiencia. Dada la naturaleza de este parámetro, la tasa de aciertos no resulta afectada (recordemos que se trata del número de hipótesis que se expanden en cada iteración, lo cual sólo tiene sentido para algoritmos de múltiples pilas sin *umbral*).

N	segs	ErrBusq	ErrModel	Aciertos	WER	PER
10	16.09	19.06	42.81	38.13	12.16	11.05
20	16.19	19.06	42.81	38.13	12.16	11.05
30	16.56	19.06	42.81	38.13	12.16	11.05
40	17.15	19.06	42.81	38.13	12.16	11.05
50	17.79	19.06	42.81	38.13	12.16	11.05
60	18.37	19.06	42.81	38.13	12.16	11.05
70	19.09	19.06	42.81	38.13	12.16	11.05
80	19.80	19.06	42.81	38.13	12.16	11.05
90	20.72	19.06	42.81	38.13	12.16	11.05
100	21.55	19.06	42.81	38.13	12.16	11.05

Tabla 6.7: Influencia del número de hipótesis a expandir por iteración (parámetro N).

Podemos ver que existe un ligero aumento del coste por frase al aumentar el valor de N. Esto es debido a que hemos utilizado heurísticos en la búsqueda. Si no hubiéramos utilizado heurísticos las diferencias en los tiempos de ejecución serían despreciables, pues en este caso, cada vez que se expande una hipótesis, el resultado de la expansión producirá una serie de hipótesis cuya *puntuación* sería peor que el de las N primeras hipótesis que se seleccionaron para la expansión, ya que las nuevas hipótesis tienen seguramente más palabras origen cubiertas, de manera que el algoritmo estaría funcionando exactamente igual que si N fuera igual a 1.

Pruebas relacionadas con la operación $addZFert(\cdot)$

En la tabla 6.8 se muestran los resultados de traducción para distintas pruebas realizadas con la operación $addZFert(\cdot)$. En ella se muestran los resultados sin realizar optimización alguna a la operación, utilizando la optimización heurística y la optimización pospuesta no heurística.

Obviamente, al no usar optimizaciones obtenemos el mayor coste por frase. Si por el contrario, usamos la optimización heurística, aumenta la eficiencia del algoritmo de manera espectacular, pero se introducen errores de búsqueda. Por último, la optimización pospuesta de $addZFert(\cdot)$ viene a ofrecer un compromiso entre las dos estrategias anteriores: no se introducen errores de búsqueda

Opt.	secs	ErrBusq	ErrModel	Aciertos	WER	PER
Sin opt.	10.63	29.43	37.79	32.78	12.75	11.57
Opt. Heur.	1.16	35.12	35.79	29.10	13.82	12.57
Opt. Post.	8.01	29.43	37.79	32.78	12.75	11.57

Tabla 6.8: Pruebas con el uso de las optimizaciones para la operación $addZFert(\cdot)$.

porque no estamos usando heurísticos, y el tiempo de traducción por frase se ve decrementado, aunque no de forma tan drástica como en el caso anterior.

Pruebas relacionadas con las operaciones $add2ZFert(\cdot)$ y $reverseAddZFert(\cdot)$

La tabla 6.9 muestra la eficiencia y el impacto en el modelo de error de las diferentes propuestas para resolver el problema de la aparición de dos palabras de fertilidad cero seguidas, y en la tabla 6.10 las tasas de error de los diferentes experimentos.

Opt.	secs	ErrInh.	ErrBusq	ErrModel	Aciertos
$addZFert$	8.01	13.04	29.43	37.79	32.78
$+add2ZFert$	17.47	2.68	19.06	42.81	38.13
$+reverseAddZFert$	41.06	8.03	24.41	39.13	36.45

Tabla 6.9: Pruebas con el uso de las operaciones $add2ZFert(\cdot)$ y $reverseAddZFert(\cdot)$ (tiempo de ejecución y modelo de error).

Opt.	WER	PER
$addZFert$	12.75	11.57
$+add2ZFert$	12.16	11.05
$+reverseAddZFert$	11.95	10.84

Tabla 6.10: Pruebas con el uso de las operaciones $add2ZFert(\cdot)$ y $reverseAddZFert(\cdot)$ (tasas de error).

Como puede apreciarse, si no intentamos resolver este problema, tenemos un buen tiempo de ejecución y mala calidad de la traducción. Si se añade el uso de la operación $add2ZFert(\cdot)$ mejora mucho la traducción pero el tiempo de ejecución sube, a pesar de que se usa la optimización heurística para $add2ZFert(\cdot)$. Por último, el peor tiempo de ejecución lo tenemos cuando además de $addZFert(\cdot)$ usamos la operación $reverseAddZFert(\cdot)$ ya que esta operación no se puede

optimizar, pero precisamente por eso, se alcanza también la mayor calidad de la traducción. Esta mejora es en cuanto a las tasas de error se refiere, no en cuanto al modelo de error (a la vista de la tabla 6.9) como podríamos pensar en un principio. Esto es debido al parámetro S (número máximo de hipótesis a almacenar en la pila), pues con la operación $reverseAddZFert(\cdot)$ la cantidad de hipótesis que se generan en el proceso de expansión es tremendamente mayor que para $add2ZFert(\cdot)$. De hecho hemos observado que, para este experimento, el número de hipótesis descartadas debido al parámetro S es muy grande para el caso de $reverseAddZFert(\cdot)$ y cero para el caso de $add2ZFert(\cdot)$. Si repitiéramos este experimento estableciendo un valor para S de modo que en ningún caso se descarten hipótesis por este motivo, el número de errores de búsqueda para el caso de la operación $reverseAddZFert(\cdot)$ sería menor que para el caso de la operación $add2ZFert(\cdot)$.

En la misma tabla podemos ver también los errores inherentes del algoritmo asociados, siendo menor cuando se utilizan las operaciones adicionales, obteniéndose el menor número cuando se usa la operación $add2ZFert(\cdot)$, también por el efecto comentado en el párrafo anterior.

Prueba relacionada con la recombinación de hipótesis

En la tabla 6.11 se muestra el impacto del uso de la recombinación en la eficiencia y tasas de error del algoritmo. Recordamos que la recombinación consiste en que dos hipótesis que no pueden distinguirse a través de los estados del modelo de lenguaje y del modelo de traducción se pueden recombinar, siempre quedándonos con aquella que tenga mayor *puntuación*.

Rec.	secs	ErrBusq	ErrModel	Aciertos	WER	PER
Si	16.09	19.06	42.81	38.13	12.16	11.05
No	48.19	19.06	42.81	38.13	12.16	11.05

Tabla 6.11: Influencia de la recombinación.

Como era de esperar, la recombinación, dado que es una optimización no heurística, no afecta a la calidad de la traducción, y en cambio si que tiene una influencia espectacular en la eficiencia.

6.6.2 Estudio empírico de eficiencia

Vamos a continuar nuestro estudio usando un traductor con las mismas características básicas que se usaron para el establecimiento de los parámetros.

Determinación de la influencia del algoritmo

En la tabla 6.12 se muestra la influencia del algoritmo en la calidad y eficiencia de la traducción.

Algoritmo	**segs**	**ErrBusq**	**ErrModel**	**Aciertos**	**WER**	**PER**
$A^\star$	17.47	19.06	42.81	38.13	12.16	11.05
Multi pila	16.10	19.06	42.81	38.13	12.16	11.05
MP con *umbral*	280.09	19.06	42.81	38.13	12.16	11.05

Tabla 6.12: Influencia del algoritmo de pila utilizado.

De entrada, y como cabía esperar, vemos que los tres algoritmos obtienen la misma tasa de aciertos y errores. Esto es así debido a que todos trabajan en esencia de la misma forma, ahora bien, el algoritmo con *umbral* resulta de largo el peor algoritmo de los tres. Esto se debe a que no hemos conseguido dar con una función de *umbral* adecuada. De hecho, el que se ha elegido consiste sencillamente en expandir la mejor hipótesis de cada cola, simplemente porque era el menos malo de todos. El problema de que tiene lugar con el algoritmo con *umbral* consiste en que se nos va de las manos el número de hipótesis a expandir en cada iteración, incluso aunque sea la cima de cada una de las pilas. Esta circunstancia se parece a aumentar enormemente el parámetro N en un algoritmo sin *umbral*, sólo que en este caso además ni siquiera se expanden las mejores hipótesis.

La eficiencia de los algoritmos de múltiples pilas y de una única pila es muy parecida, aunque ligeramente mejor para el primero. La razón de ello reside en la limitación del tamaño de las pilas. Para el algoritmo de múltiples pilas hemos utilizado un valor de S igual a $1,000$, mientras que para el algoritmo de una sola pila es de $100,000$, lo que reduce el coste por inserción de hipótesis en la pila significativamente. No obstante el algoritmo de múltiples pilas se degradará a medida que se incrementa el tamaño de la frase, debido al término exponencial que aparece en la expresión de la complejidad. Para ilustrar esto, en la tabla 6.13 se muestra, para el caso del algoritmo de única pila, el coste temporal del proceso de expansión de hipótesis con respecto al coste total de la ejecución del algoritmo. Como se puede apreciar en la tabla, el coste medio relativo por expansión es prácticamente el 100% del tiempo del algoritmo, aspecto deseable pues eso significa que el tiempo de ejecución se gasta precisamente en eso, en expandir hipótesis.

En la tabla 6.14 se muestran los mismos datos para el algoritmo de múltiples pilas. La diferencia es bastante significativa, sobre todo con frases de tamaño 12 con las cuales el algoritmo de múltiples pilas empieza a invertir una cantidad de tiempo apreciable en seleccionar las hipótesis a expandir; esta, por contra, es una característica tremendamente negativa.

Long. frase	**4**	**6**	**8**	**10**	**12**
Coste relativo medio(%)	99.1	99.5	99.4	99.5	99.6
Mínimo coste relativo(%)	97.1	97.9	95.5	93.8	96.0

Tabla 6.13: Coste relativo de la expansión para un algoritmo con una sola pila.

Long. frase	**4**	**6**	**8**	**10**	**12**
Coste relativo medio(%)	98.3	98.8	98.5	97.9	94.3
Mínimo coste relativo(%)	96.4	96.9	94.5	93.1	**82.5**

Tabla 6.14: Coste relativo de la expansión para un algoritmo con múltiples pilas.

Determinación de la influencia del modelo utilizado.

En la tabla 6.15 se muestra la influencia del modelo utilizado en el coste temporal del algoritmo y su tasa de aciertos, las diferencias son sustanciosas en ambos casos: El modelo 3 tiene el inconveniente de que no es posible realizar una optimización no heurística de la operación $addZFert(\cdot)$, de manera que no ha sido utilizada para hacer el experimento. Por otro lado, en su historia generativa veíamos que se contemplan todas las permutaciones posibles para las palabras con fertilidad mayor que 1, de manera que el algoritmo es bastante más complejo. Por otro lado, la tasa de aciertos del algoritmo es bastante mejor en el modelo 3, esto se debe a que la optimización para $add2ZFert(\cdot)$ utilizada es ligeramente distinta, ya que tiene en cuenta el parámetro de distorsión, por lo que se descartan menos operaciones de este tipo. Otro error que aparece es específico del modelo 4 y relacionado con el parámetro W, que ilustramos en la figura 6.14. La palabra *fiftenth* debe alinearse primero con la palabra *día* y después realizar una extensión con la palabra *quince* si usamos el modelo 4. Sin embargo *fiftenth* no está entre las W mejores traducciones de *día* y sí *eighth*. En el modelo 3 era posible alinear primero con *quince* y después extender a *día* con lo que en este caso sí que se podía alcanzar la traducción óptima. Este problema se resuelve aumentando el parámetro W.

Por otro lado, aparecen más errores del modelo, la causa de esto podría estar en el entrenamiento del mismo; en cualquier caso, se requeriría un estudio más detallado de por qué se ha producido este resultado.

Determinación de la influencia del heurístico.

En la tabla 6.16 se muestra la influencia del heurístico en la búsqueda. Recordamos que el cálculo de un heurístico consiste en la estimación del coste de completar una hipótesis parcial.

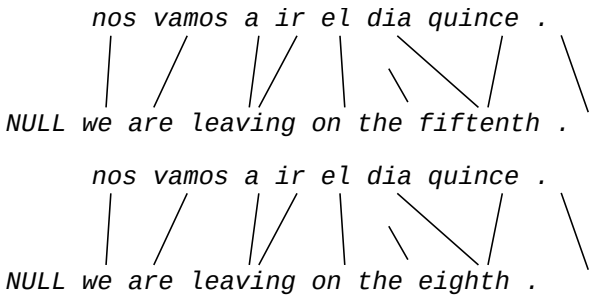


Figura 6.14: Ejemplo de error obtenido con el modelo 4 relacionado con el parámetro W.

Mod.	segs	ErrBusq	ErrModel	Aciertos	WER	PER
3	104.22	13.38	45.48	41.14	12.43	10.77
4	48.33	19.06	42.81	38.13	12.16	11.05

Tabla 6.15: Influencia del modelo de traducción utilizado.

Como ya hemos adelantado, al ser heurísticos admisibles no introducen errores de búsqueda adicionales con lo que no modifican los resultados obtenidos, aunque, sí podemos apreciar una reducción considerable en el tiempo de ejecución, llegando, en el mejor de los casos, a reducirse a la mitad el tiempo de ejecución.

heur.	segs	ErrBusq	ErrModel	Aciertos	WER	PER
Ning.	23.22	19.06	37.46	43.48	10.42	9.73
T	16.67	19.06	37.46	43.48	10.42	9.73
TF	16.83	19.06	37.46	43.48	10.42	9.73
TFL	12.13	19.06	37.46	43.48	10.42	9.73

Tabla 6.16: Influencia del heurístico utilizado.

Como podemos ver en la tabla 6.16, el heurístico T introduce una notable mejora en la eficiencia, mientras que TF no avanza nada con respecto a su predecesor. Para explicar esto podemos fijarnos en fuerza predictiva de ambos mostrada en la tabla 6.17. Como podemos ver en dicha tabla la fuerza predictiva de ambos heurísticos es prácticamente idéntica, dado que la puntuación que se asigna a la hipótesis vacía según ellos es muy parecida. Con diferencia el mejor heurístico es el TFL reduciendo en más de la mitad el tiempo de ejecución comparando con la no utilización de heurístico alguno, aunque requiere una fase inicial de precálculo de las probabilidades del modelo de lenguaje prohibitiva en tiempo si se toman corpus con elevado número de palabras. Del mismo modo,

y como era de esperar, este heurístico presenta la mejor fuerza predictiva de los tres.

heur.	Punt. óptima	Punt. heur	Fuerza predictiva
T	28.90	5.70	23.20
TF	28.90	5.89	23.01
TFL	28.90	10.25	18.65

Tabla 6.17: Fuerza predictiva del heurístico utilizado.

Determinación de la influencia del tamaño de frase.

Para este experimento hemos elegido distintos conjuntos de test (de la tarea EuTrans-I), de distintas longitudes, concretamente 5 conjuntos de 100 frases de test cada uno, de longitudes 4, 6, 8, 10 y 12 palabras, respectivamente. En la tabla 6.18 podemos ver la información del modelo de error y tiempo medio de traducción de las frases, donde se hace patente una influencia muy estrecha entre la longitud de frase y el coste temporal del algoritmo. En la misma tabla se muestran las tasas de error para este experimento, donde podemos ver que esa diferencia no parece estar tan clara en lo que a la tasa de aciertos se refiere.

Long.	secs	ErrBusq	ErrMode	Aciertos	WER	PER
4	0.27	20	40	40	34.57	33.33
6	1.53	13	47	40	17.49	16.80
8	11.76	20	44	36	11.87	11.19
10	26.74	9	46	45	7.54	7.45
12	165.59	17	51	32	11.34	11.01
media	49.56	14.94	46.75	38.31	12.87	12.40

Tabla 6.18: Influencia de la longitud de frase.

6.6.3 Elección del algoritmo

Vistos los resultados obtenidos en la experimentación llevada a cabo en esta sección, e intentando establecer un compromiso entre eficiencia y aciertos en la traducción, podemos concluir que una buena elección de un traductor será aquel que utilice un algoritmo de una única pila, que use el modelo 4 como modelo de traducción, optimizado para la operación $addZFert$ con la optimización pospuesta, y un heurístico, no necesariamente el TFL por los problemas que conlleva su cálculo (aunque obtiene los mejores resultados). Será importante escoger un buen valor de Z, por ejemplo 25 ya que este parámetro ha demostrado

tener un gran impacto en eficiencia y la tasa de aciertos de los algoritmos. El valor del parámetro A puede fijarse en 6, por ser el valor por encima del cual ya no se produjo la mejora. En cuanto al parámetro W hemos visto que un valor de 20 sería aceptable. Del mismo modo establecemos el valor idóneo de S a 1000.

6.7 Resultados.

Vamos a llevar a cabo una serie de experimentos utilizando un traductor con los parámetros y características establecidos en la sección anterior. Para hacer dichos experimentos se han escogido dos tareas distintas: EuTrans-I, y Hansards, cuyas características fueron descritas en el capítulo 2, sección 2.7. Concretamente utilizaremos los siguientes conjuntos de test: 2636 frases para la tarea EuTrans-I, aquellas de longitud menor o igual que 15 y 500 frases para la tarea Hansards, de longitudes que varían de 4 a 12 palabras por frase.

6.7.1 Resultados con la tarea EuTrans-I

En la tabla 6.19 podemos ver la información del modelo de error y tiempo medio de traducción de las frases. En la misma tabla se muestran las tasas de error para este experimento. Para este experimento hemos utilizado el modelo 4, como heurístico el TFL, y hemos fijado los siguientes valores de los parámetros de optimización: $A = 6$, $W = 20$, $Z = 25$, $N = 10$ y $S = 1000$.

secs	InhErr	ErrBus	ErrModel	Aciertos	WER	PER
87.12	2.12	18.44	44.08	37.48	14.21	11.13

Tabla 6.19: Resultados con la tarea EuTrans-I para los algoritmos de pila. Modelo y tasas de error.

A la vista de los resultados podemos ver que con la configuración establecida el número de errores inherentes a algoritmo no es cero, concretamente se obtiene el 2.12% de errores de este tipo. Una vez más cabe destacar la gran cantidad de errores del modelo que se obtienen. En cuanto a las tasas de error podemos ver que se obtienen unos valores razonables y de la misma magnitud que para los algoritmos basados en programación dinámica, aunque los tiempos de ejecución son sustancialmente más altos.

6.7.2 Resultados con la tarea Hansards

Al igual que en la experimentación llevada a cabo en el capítulo anterior para esta tarea, hemos realizado experimentos para cinco corpus de 100 frases cada uno, de frases de longitud 4, 6, 8, 10 y 12, dada la complejidad que esta tarea conlleva.

Para tamaños mayores el tiempo de computación es enorme, las causas de esto no estaban demasiado claras en un principio, pero tras realizar una experimentación similar a la presentada para la tarea EuTrans-I (ver apéndice C) han demostrado que para traducir estos corpus es necesario limitar los parámetros del algoritmo e introducir una nueva optimización.

En concreto, se hemos fijado el valor del parámetro A a 3, los de W y Z a 8, S a 10000, y como heurístico hemos utilizado el TF. Además se aplica una nueva poda para las W mejores traducciones consistente en que se borran de esa lista aquellas traducciones cuya probabilidad sea inferior a 0.01 veces la probabilidad de la mejor traducción.

Los modelos de traducción y lenguaje utilizados en este experimento han sido los mismos que los que se utilizaron para los algoritmo basados en programación dinámica (ver sección 5.9.2, página 134). Simplemente recordar que se ha utilizado un corpus de 128,000 pares de frases de entrenamiento, siguiendo el esquema de entrenamiento $1^5 2^5 3^5 4^5 5^5$.

En la tabla 6.20 podemos ver la información del modelo de error y el tiempo medio de traducción de las frases. En la misma tabla se muestran las tasas de error para este experimento, donde se presentan resultados para los distintos corpus utilizados, de modo que también se puede ver la influencia de la longitud de la frase en el comportamiento del algoritmo.

Long.	**segs**	**ErrBusq**	**ModelErr**	**Aciertos**	**WER**	**PER**
4	0.88	25	38	37	40.65	40.40
6	19.63	12	81	7	51.10	49.92
8	100.80	13	85	2	54.92	51.99
10	247.87	6	93	1	59.34	55.95
12	446.45	4	96	0	64.77	58.23
media	163.13	12	78.60	9.40	54.16	51.30

Tabla 6.20: Influencia de la longitud de frase en el modelo y tasas de error para la tarea Hansards, para los algoritmos de pila.

Los resultados son bastante peores que para la tarea de EuTrans-I, sin embargo, en [Och et al. 01] se da un valor de *WER* para el corpus de Hansards con frases de cuatro, seis, ocho, diez y doce palabras en torno a 73, lo que hace pensar que no estamos demasiado lejos de los resultados con otras implementaciones, es más, los resultados aquí obtenidos son mucho mejores que los allí presentados.

Podemos concluir en definitiva que esta tarea es muchísimo más compleja que la del turista y que nuestros algoritmos de pila no son capaces de resolverla adecuadamente.

6.8 Conclusiones

En este capítulo hemos desarrollado todos los algoritmos de pila existentes en la bibliografía. A pesar de que muchos de ellos ya han sido desarrollados previamente se echaba en falta una puesta en común de todos ellos, de las diversas optimizaciones en el proceso de búsqueda que diversos autores proponen, así como implementar todas sus versiones para los modelos 3 y 4. Además se han propuesto diferentes soluciones a ciertos errores de búsqueda inherentes a los algoritmos de pila. Otro punto poco estudiado en la bibliografía ha sido el efecto que las distintas optimizaciones en cuanto a eficiencia y eficacia. Nosotros hemos realizado un estudio minucioso de ello.

Por otra parte hemos estudiado su complejidad teórica, aspecto poco tratado en la bibliografía relacionada con el tema.

Como aportaciones más importantes de este capítulo podemos destacar:

- Los algoritmos de pila, bajo determinadas circunstancias, son capaces de encontrar la solución óptima al problema de la búsqueda.
- Hemos puesto de manifiesto una característica poco deseable de los algoritmos de múltiples pilas reflejada en el término de exponencial que aparece en la expresión de su complejidad por iteración, y que equivale a decir que, para valores grandes de J, estos algoritmos pasan más tiempo buscando las mejores hipótesis a expandir que expandiéndolas, característica que les resta mucho interés.
- En cuanto a los algoritmos basados en pila en su conjunto, hemos visto que tienen limitaciones importantísimas en lo que respecta a su coste temporal, incluso con frases cortas como las que hemos manejado.
- Hemos introducido nuevas operaciones para solventar errores inherentes de estos algoritmos aunque por contra hacen todavía más lenta la traducción.

A pesar de todo cabe matizar, como ya hemos comentado, que en [Germann et al. 01] se da un tiempo de traducción medio para frases de longitud 15 superior a 30 minutos, de modo que los tiempos de traducción aquí obtenidos no son del todo malos.

No obstante, la utilización de estos traductores, según comentan algunos autores como [Och et al. 01], pasaría por ser usados como una herramienta de depuración de otras herramientas de traducción con tiempos de ejecución mucho más razonables. Otra posibilidad de uso, es la de detectar errores del modelo de traducción, ya que se han obtenido tasas muy altas de errores de este tipo.

En vistas al futuro, sería interesante investigar un poco más sobre el uso de *umbrales* en los algoritmos, a fin de determinar las mejoras que puedan aportar, quizás el tipo de algoritmo de pila menos estudiado en este trabajo. También sería interesante estudiar una forma de resolver el problema de los errores inherentes al algoritmo, que en principio, parece que nos obligaría a redefinir el

algoritmo de expansión. La forma de hacer esto adecuadamente no está nada clara.

Evidentemente, también entra dentro de nuestros planes adaptar todos estos algoritmos para los modelos HMM y 5.

Capítulo 7

Algoritmos de búsqueda voraces

En este capítulo expondremos la última familia de algoritmos de búsqueda desarrollados en esta tesis. Estos algoritmos siguen el paradigma de la técnica devoradora para resolución de problemas de búsqueda. En general, los algoritmos expuestos en este capítulo se basan en métodos de optimización local, es decir a partir de una solución inicial se intentará mejorarla aplicando el método de ascenso de colina, el cual se basa en construir un vecindario de posibles soluciones (aplicando una serie operaciones a la hipótesis actual), elegir la mejor de acuerdo a un determinado criterio e iterar el proceso hasta que no se pueda mejorar.

7.1 Introducción

La gran diferencia de los algoritmos que expondremos a continuación con respecto a los propuestos hasta ahora, radica en que en vez de seguir un proceso incremental de construcción de hipótesis parciales, se establece una hipótesis completa inicial, y un alineamiento asociado, sobre la cual se realizarán ciertas operaciones o transformaciones que la irán modificando y mejorando iterativamente de acuerdo a un determinado criterio.

En este caso, al igual que para los algoritmos expuestos en el capítulo anterior, el criterio de búsqueda a resolver será el que se comento en la expresión 6.2, que

es:

$$\hat{\mathbf{e}} \approx \arg\max_{\langle \mathbf{e},\mathbf{a} \rangle} \{Pr(\mathbf{e}) \cdot Pr(\mathbf{f},\mathbf{a} \mid \mathbf{e})\} \tag{7.1}$$

Es decir, se trata de, a partir de una hipótesis y un alineamiento iniciales, ir obteniendo iterativamente mejores hipótesis de salida junto con los alineamientos asociados con respecto a la frase de entrada.

A pesar de que los algoritmos voraces obtienen la solución óptima para algunos problemas clásicos, como por ejemplo el de encontrar el camino mínimo en un grafo o el problema de la mochila fraccionada, no lo es en este caso. En el caso que nos compete podríamos decir que la solución propuesta en este capítulo sigue una aproximación algo más genérica pues, en definitiva, se trata de la aplicación de métodos de optimización local siguiendo una aproximación voraz dado que una vez tomada una decisión en una iteración se mantiene hasta el final del proceso de búsqueda.

Dada la naturaleza de los métodos de optimización local los algoritmos que expondremos a continuación no podrán garantizar la optimalidad de la solución. Estos métodos son muy sensibles a la inicialización de manera que solo se puede garantizar la obtención de óptimos locales. Este problema se muestra gráficamente en la figura 7.1.

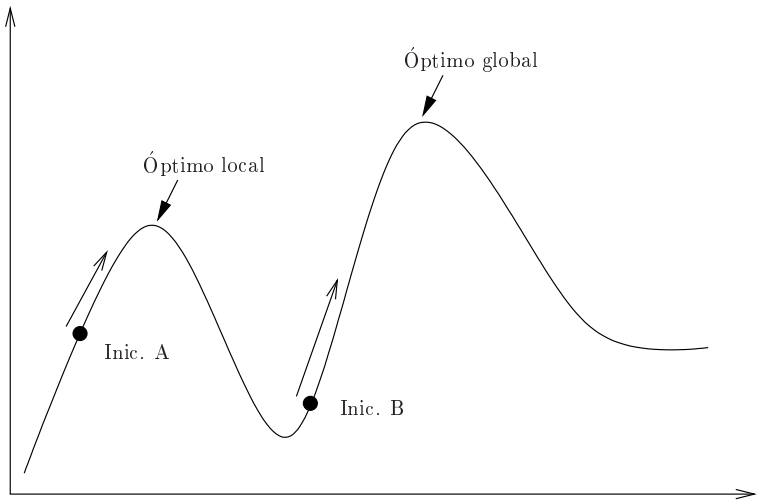


Figura 7.1: Problema de la inicialización del método *hillclimbing*. Empezando en A se alcanza un óptimo local, mientras que empezando en B se alcanza el global, a pesar de que A es mejor inicialización que B de acuerdo a la función objetivo.

A pesar de esta gran limitación, y como veremos en la sección de resultados, estos algoritmos poseen algunas características que los hacen atractivos:

- Son muy rápidos.
- Se pueden utilizar como etapa de refinamiento a otros algoritmos de búsqueda.
- Existen infinidad de técnicas de optimización local basadas en este paradigma, las cuales se pueden aplicar fácilmente y con poco esfuerzo a los algoritmos presentados.

7.2 Algoritmo básico *GreedySearch*

Los algoritmos que expondremos en este capítulo están basados en un algoritmo voraz propuesto en [Germann et al. 01] para el modelo 4, adaptados convenientemente al resto de modelos de IBM y al modelo HMM.

Dado el criterio de búsqueda a resolver, la aplicación de este tipo de algoritmos a cualquier modelo de alineamiento M pasa por poder calcular de forma eficiente el término $Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$. En nuestro caso resolver $Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$ consiste en calcular esa probabilidad para el modelo concreto a utilizar, lo cual será relativamente sencillo dado que, según el planteamiento expuesto, siempre serán completamente conocidos los valores de $\mathbf{e}$ y $\mathbf{a}$ para una hipótesis en concreto. Esto es así dado que en todo momento trabajaremos con hipótesis completas, por contra de los algoritmos presentados en capítulos anteriores.

En el algoritmo 7.1 podemos ver cómo quedaría el algoritmo básico *GreedySearch*, el cual sigue el siguiente esquema de funcionamiento:

1. Obtener una hipótesis inicial $h = \langle \mathbf{e}, \mathbf{a} \rangle$ (función $InitGreedy(f_1^J)$ en el algoritmo).
2. Iterar:
   (a) Aplicar transformaciones a $\mathbf{a}$, para obtener un conjunto de vecinos $\mathcal{N}(\mathbf{a})$ (función $ComputeNeighborhood(\mathbf{a})$ en el algoritmo). Cada vecino de un par $\langle \mathbf{e}, \mathbf{a} \rangle$, o elemento de dicho conjunto, será aquel que difiera de éste en exactamente el resultado de la aplicación de una y solo una transformación.
   (b) Sea h' la mejor hipótesis de $\mathcal{N}(\mathbf{a})$ de acuerdo a la probabilidad $Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$ (según el modelo M).
   (c) Si h' no mejora a h parar y devolver h, sino, si se ha alcanzado un número máximo de iteraciones parar y devolver h'.

Según se ha planteado el algoritmo 7.1, éste requerirá almacenar el conjunto $\mathcal{N}(\mathbf{a})$ y posteriormente recorrerlo elemento a elemento para calcular su *puntuación* o probabilidad $Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$ asociada. Este funcionamiento tal y como se plantea es bastante ineficiente pues de algún modo se podría evaluar cada elemento del conjunto $\mathcal{N}(\mathbf{a})$ en el momento de su creación, quedándonos en todo

Algoritmo: ***GreedySearch()***
entrada: $\mathbf{f}, maxIter, M$
salida: $\langle \mathbf{e}, \mathbf{a} \rangle$
  $//Inicialización$
  $\langle \mathbf{e}, \mathbf{a} \rangle = InitGreedy(f_1^J)$
  $score = Pr(\mathbf{e}) \cdot Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$; $bestHip = \langle \mathbf{e}, \mathbf{a} \rangle$
  $//Iteración$
  $iter = 0$
  repetir
    $newScore = score$
    $\mathcal{N}(\mathbf{a}) = ComputeNeighborhood(\mathbf{a})$
    para_todo $\langle \mathbf{e}, \mathbf{a} \rangle \in \mathcal{N}(\mathbf{a})$ hacer
      $currScore = (Pr(\mathbf{e}) \cdot Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e}))$
      si $score < currScore$ entonces
        $score = currScore$; $bestHip = \langle \mathbf{e}, \mathbf{a} \rangle$
      fin_si
    fin_para
    $iter = iter + 1$
  hasta $newScore < score$ and $iter < maxIter$
  devolver: $bestHip$

Algoritmo 7.1: Algoritmo voraz básico *GreedySearch*.

momento con el mejor, lo cual reduciría considerablemente los requerimientos espaciales del algoritmo y además evitaría tener que recorrer a posteriori el conjunto $\mathcal{N}(\mathbf{a})$. Volveremos a este tema en la sección 7.3.

En definitiva los ingredientes básicos de un algoritmo voraz serán:

- Modelos estadísticos de traducción y lenguaje. Al igual que en casos anteriores utilizaremos trigramas como modelo de lenguaje. En cuanto al modelo de traducción podremos utilizar cualquiera que nos permita calcular la probabilidad $Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$. En nuestro caso cualquiera de los expuestos en el capítulo 3.
- Mecanismo de inicialización.
- Algoritmo para construir $\mathcal{N}(\mathbf{a})$ a partir de un alineamiento $\mathbf{a}$ dado y, por tanto, definición de las operaciones/transformaciones asociadas.

En las siguientes secciones veremos con todo detalle los dos últimos.

7.2.1 Inicialización de hipótesis

Cómo hemos comentado anteriormente la inicialización es un punto crítico en el diseño de este tipo de algoritmos. Nosotros utilizaremos dos posibles inicializaciones:

- La propuesta en [Germann et al. 01] consistente en definir cómo hipótesis inicial la formada por: la mejor traducción inversa de cada palabra de la frase de entrada f_1^J, y el alineamiento $\mathbf{a} = a_1^J$, completamente monótono con la frase de entrada. Esta inicialización la podemos ver en el algoritmo 7.3 con más detalle. Como podemos ver en el algoritmo, el número de palabras de la hipótesis inicial tendrá el mismo número de palabras que la frase de entrada, es decir J, siempre y cuando la mejor traducción inversa de una palabra f_j en concreto no sea la palabra vacía $NULL$.

- Otra posible inicialización, aquí propuesta, cambia con respecto a la anterior en cuanto al alineamiento inicial se refiere. Una vez se han generado las palabras a formar parte de la hipótesis se realiza un alineamiento por Viterbi de acuerdo al modelo de traducción a utilizar. De forma algorítmica lo podemos ver en el algoritmo 7.3.

Este segundo tipo de inicialización servirá de ayuda en el caso en que dos o más palabras tengan traducciones parecidas, con lo que el alineamiento por Viterbi puede realizar una mejor inicialización que será distinta del alineamiento monótono. De cualquier modo al ser un alineamiento por Viterbi, siempre tendrá una mayor *puntuación* que el propuesto en [Germann et al. 01]. En la implementación, el parámetro que indica si se inicializa de este modo, provocará también que tras cada iteración del algoritmo se recalcule el alineamiento obtenido según Viterbi, con lo que la *puntuación* siempre será igual o mejor al obtenido previamente.

En la figura 7.2 podemos ver una inicialización para una frase real del corpus EuTrans-I.

Algoritmo: ***InitGreedySearch()***
entrada: $I, t(\cdot), a(\cdot), f_1^J$
salida: $\langle \mathbf{e}, \mathbf{a} \rangle$
  para $j = 1 \cdots J$ hacer
    $i = 1$
    $e = \arg\max_e t^{-1}(e \mid f_j)$
    si $e <> NULL$ entonces
      $e_i = e$
      $a_j = i$
      $i = i + 1$
    sino
      $a_j = 0$
    fin_si
  fin_para
  $I = i - 1$
  devolver: $\langle e_1^I, a_1^J \rangle$

Algoritmo 7.2: Inicialización al algoritmo *GreedySearch()*.

Algoritmo: ***InitGreedySearchVit()***
entrada: M, f_1^J
salida: $\langle \mathbf{e}, \mathbf{a} \rangle$
  para $j = 1 \cdots J$ hacer
    $i = 1$
    $e = \arg\max_e t^{-1}(e \mid f_j)$
    si $e <> NULL$ entonces
      $e_i = e$
      $i = i + 1$
    fin_si
  fin_para
  $I = i - 1$
  $a_1^J = V_M(e_1^I, f_1^J)$
  devolver: $\langle e_1^I, a_1^J \rangle$

Algoritmo 7.3: Inicialización por Viterbi al algoritmo *GreedySearch()*.

7.2.2 Algoritmo de construcción de $\mathcal{N}(\mathbf{a})$

Una vez se ha establecido un alineamiento inicial $\mathbf{a}$, e implícitamente una hipótesis inicial $\mathbf{e}$, el algoritmo voraz deberá crear el conjunto de vecinos $\mathcal{N}(\mathbf{a})$ para intentar buscar un alineamiento de mayor probabilidad $Pr(\mathbf{e}) \cdot Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$. Para ello definimos las siguientes operaciones (o transformaciones) a aplicar a un alineamiento dado[1]:

swapSegments(i, i', k, k')**:** Esta operación crea un nuevo alineamiento intercambiando los segmentos $e_i^{i'}$ y $e_k^{k'}$. Los segmentos no se podrán solapar, es decir se deber cumplir que $i' < k$, y evidentemente que $i \leq i'$ y $k \leq k'$. La aplicación de esta operación no cambiará el alineamiento, es decir se preservarán las conexiones de la hipótesis actual. Además se deberá cumplir que $(i' - i + 1) + (k' - k + 1) \leq I$, siendo I la talla de la hipótesis actual. En definitiva los segmentos podrán estar formados por una palabra o a lo sumo por $I - 1$ palabras.

Adicionalmente hemos definido la operación $swapSegmentsWithAlig(\cdot)$, que es idéntica a la anterior con la excepción de que el alineamiento si se verá modificado con el intercambio.

translateOneOrTwoWords(j, e, j', e')**:** Esta operación cambiará la traducción de una o dos palabras origen, las cuales ocupen las posiciones j y j', de e_{a_j} y $e_{a_{j'}}$ a e y e', respectivamente. Se podrán dar las siguientes situaciones especiales en particular:

[1] Seguiremos la terminología utilizada en [Germann et al. 01]

- Que e_{a_j} (o $e_{a_{j'}}$) sea una palabra de fertilidad igual a 1, y que e (o e') sea la palabra vacía ($NULL$), entonces e_{a_j} (o $e_{a_{j'}}$) se borrará de la hipótesis. Es decir, si la nueva traducción de una palabra origen pasa a ser la palabra vacía, entonces la palabra destino que hasta la fecha daba lugar a ella tendrá que desaparecer de la traducción. En este caso la palabra a insertar será una palabra de fertilidad 0.
- Que e_{a_j} (o $e_{a_{j'}}$) sea $NULL$, entonces e (o e') se insertará en la posición que produzca el alineamiento de mayor probabilidad. Este caso es justo el contrario al anterior.
- Que $e_{a_j} = e$ o que $e_{a_{j'}} = e'$, en este caso la operación solo se afectará a una sola palabra, aquella que sea distinta de su nueva traducción.

$translateAndInsertZFert(j, e, ze)$**:** Esta operación cambia la traducción de la palabra origen que ocupa la posición j por la palabra e, es decir cambia e_{a_j} por e, y simultáneamente inserta la palabra ze en la posición que produzca el alineamiento de mayor probabilidad. Evidentemente, ze será una palabra de fertilidad 0. En este caso, al igual que en el anterior, si $e_{a_j} = e$ entonces la operación se limitará a insertar una palabra de fertilidad 0 en la mejor posición dentro de la hipótesis.

$joinWords(i, i')$**:** Esta operación elimina de la hipótesis la palabra de la posición i (o la palabra i') y alinea las palabras de $\mathbf{f}$ alineadas con e_i (o $e_{i'}$) con la palabra i' (o i).

$removeZFert(i)$**:** Esta operación elimina la palabra de fertilidad 0 que ocupa la posición i en el alineamiento actual.

El haber elegido estas operaciones, y no otras que nos podrían ocurrir, se debe principalmente a dos razones:

- Son lo suficientemente generales para posibilitar al traductor eludir máximos locales (un gran problema de estos algoritmos) y modifican de forma no trivial un alineamiento dado para intentar obtener buenas traducciones.
- Por otra parte, no son excesivamente costosas desde el punto de vista del tiempo de ejecución.

En el algoritmo 7.4 se muestra cómo combinar estas operaciones para obtener un vecindario $\mathcal{N}(\mathbf{a})$ para un alineamiento $\mathbf{a}$ dado. El algoritmo funciona aplicando, sucesivamente las operaciones definidas a un alineamiento dado $\mathbf{a}$ y la correspondiente hipótesis $\mathbf{e}$ asociada. Al final habremos construido un vecindario $\mathcal{N}(\mathbf{a})$ para dicho alineamiento, del cual se podrá extraer la mejor hipótesis obtenida para pasar a la siguiente iteración del algoritmo *GreedySearh*.

En la figura 7.2 podemos ver un ejemplo real de la aplicación de algunas de las operaciones definidas para obtener la traducción correcta de la frase *me gustaría*

Algoritmo: ***ComputeNeighborhood(a)***

entrada: $\mathbf{a}$
salida: $\mathcal{N}(\mathbf{a})$
  $\mathcal{N}(\mathbf{a}) = \emptyset$
  $//swapSegments$
  para $i = 1 \cdots I$ y $i' = i \cdots I$ hacer
    para $k = i' + 1 \cdots I$ y $k' = k \cdots I$ hacer
      $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{a}) \cup swapSegments(i, i', k, k')$
    fin_para
  fin_para
  $//translateOneOrTwoWords$
  para $j = 1 \cdots J$ y $j' = j + 1 \cdots J$ hacer
    para_todo $e, e' \in \mathcal{E}$ hacer
      $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{a}) \cup translateOneOrTwoWord(j, e, j', e')$
    fin_para
  fin_para
  $//translateAndInsertZFert$
  para $j = 1 \cdots J$ hacer
    para_todo $e, ze \in \mathcal{E}$ hacer
      $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{a}) \cup translateAndInsertZFert(j, e, ze)$
    fin_para
  fin_para
  $//joinWords$
  para $i = 1 \cdots I$ y $i' = i \cdots I$ hacer
    $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{a}) \cup joinWords(i, i') \cup joinWords(i', i)$
  fin_para
  $//removeZFert$
  para $i = 1 \cdots I$ hacer
    $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{a}) \cup removeZFert(i)$
  fin_para
  devolver: $\mathcal{N}(\mathbf{a})$

Algoritmo 7.4: Algoritmo para el cálculo del vecindario de un alineamiento dado.

cambiarme a una habitación más cálida . de la tarea EuTrans-I. Como podemos ver se realizan 5 iteraciones del algoritmo habiéndose aplicado las siguientes operaciones, en cada una:

Iter.1 $translateAndInsert(1, \textit{would}, \textit{to})$: cambia la traducción de la palabra *me* a *would* y además inserta la palabra de fertilidad 0 *to* en la posición 3.

Iter.2 $translateAndInsert(5, \textit{warmer}, \textit{a})$: en esta operación podemos ver un caso particular de la operación $translateAndInsert(\cdot)$, pues se da el caso de

que la palabra *otra* ya es traducción de *warmer*, con lo que dicha operación solo realiza la inserción de la palabra de fertilidad 0 *a* en la posición 6.

Iter.3 $joinWords(9, 10)$: esta operación elimina la palabra *quieter* uniendo su traducción a *warmer*.

Iter.4 $joinWords(9, 7)$: esta es idéntica a la anterior pero con las posiciones 9 y 7.

Iter.5 $translateAndInsert(1, \textit{would}, I)$: en este caso ocurre el mismo efecto que en la iteración 3, pero en este caso se inserta la palabra de fertilidad 0 *I* en la posición 1, obteniéndose así la traducción correcta de la frase.

Hay que destacar que tras cada iteración se recalcula el alineamiento obtenido según Viterbi, hecho que motiva que en algunos casos el alineamiento cambie siempre y cuando se obtenga una mejor *puntuación*. Esto también se ha aplicado al ejemplo de la figura 7.2.

7.3 Complejidad computacional

La complejidad computacional por iteración del algoritmo *GreedySearch* vendrá dada por el coste que requiere el algoritmo de construcción del conjunto de vecinos $\mathcal{N}(\mathbf{a})$ para un alineamiento dado.

A la vista del algoritmo 7.1, su coste computacional será proporcional al coste de calcular $Pr(\mathbf{e}) \cdot Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$ para cada uno de los elementos del conjunto $\mathcal{N}(\mathbf{a})$ que se genere en cada iteración. Como ya hemos anticipado en la sección 7.2 no será necesario enumerar todo el conjunto $\mathcal{N}(\mathbf{a})$, pues se puede calcular el valor de $Pr(\mathbf{e}) \cdot Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$ en el mismo momento en que se genera la hipótesis $\mathbf{e}$, con lo que se podrá descartar si no mejora el alineamiento obtenido hasta ese momento en la iteración actual. El coste de calcular $Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$, dependerá del modelo a utilizar pero en el peor de los casos será del orden de $O(I)$ (suponiendo que el coste del cálculo del factorial es constante dado que se puede precalcular), y el de $Pr(\mathbf{e})$ será del orden de $O(I)$, siendo I la longitud de la hipótesis $\mathbf{e}$. Es decir, el coste asintótico de calcular la *puntuación* de una hipótesis dada será del orden de $O(I)$.

Por tanto, la complejidad computacional del algoritmo *GreedySearch* vendrá determinada por el coste de cada una de las operaciones definidas y por el número de veces que cada una de ellas se deba realizar. Veamos el coste asociado a cada una de las operaciones observando el código del algoritmo 7.4:

$swapSegments(\cdot)$**:** El coste de intercambiar dos segmentos se puede considerar constate, con lo que su coste será de $O(I)$ debido al coste de calcular $Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$. En este caso el número de veces que se deberá ejecutar es proporcional a I^4. El coste para la operación $swapSegmentsWithAlig(\cdot)$ no sufre modificación alguna.

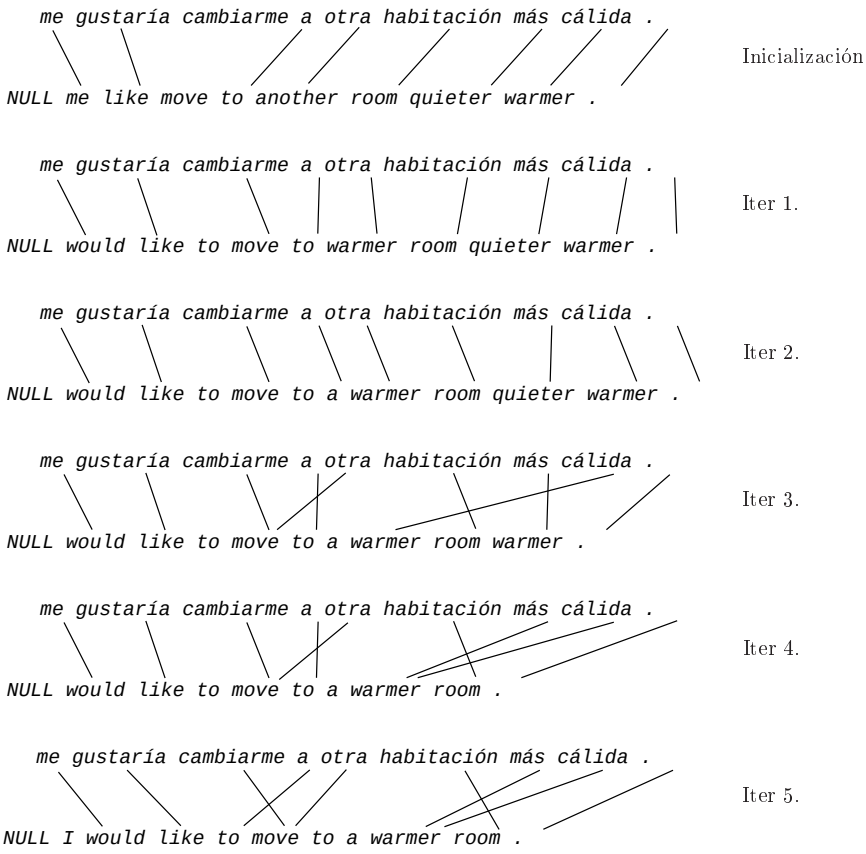


Figura 7.2: Ejemplo de aplicación de algunas de las operaciones para conseguir la traducción de una frase dada mediante el algoritmo *GreedySearch*.

$translateOneOrTwoWords(\cdot)$**:** En el peor de los casos, es decir cuando alguna de las palabras a cambiar su traducción estuviese alineada con la palabra $NULL$, esta operación requerirá recorrer toda la hipótesis posición a posición para ver donde mejor cabida tenga la nueva palabra, es decir tendrá una complejidad del orden de I multiplicado por el coste de calcular $Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$. Es decir, tendrá un coste de $O(I^2)$. Además esta operación deberá de realizarse un total de $J^2 \cdot \mid \mathcal{E} \mid^2$ veces, por los 4 bucles anidados.

$translateAndInsertZFert(\cdot)$**:** Esta operación tendrá un coste idéntico al de la operación anterior pues el hecho de insertar la palabra de fertilidad cero requiere recorrer de nuevo la hipótesis, es decir $O(I^2)$. En cambio el número de veces que esta operación deberá ejecutarse es $J \cdot \mid \mathcal{E} \mid^2$.

$joinWords(\cdot)$: Al igual que las anteriores el coste de unir dos palabras es constante, con lo que su coste volverá a ser $O(I)$. En este caso el número de veces que se deberá ejecutar es proporcional a I^2.

$removeZFert(\cdot)$**:** Esta operación tendrá un coste constante, es decir no incluye ningún factor adicional, con lo que será $O(I)$. El número de veces que deberá ejecutarse será I.

A la vista de los costes parciales referentes a cada operación, y dado que todas ellas se ejecutan de forma secuencial, podemos decir que el coste computacional del algoritmo *GreedySearch* es del orden de:

$$O(I^4 \cdot I + J^2 \cdot \mid \mathcal{E} \mid^2 \cdot I^2 + J \cdot \mid \mathcal{E} \mid^2 \cdot I^2 + I^2 \cdot I + I \cdot I) \equiv O(J^2 \cdot \mid \mathcal{E} \mid^2 \cdot I^2)$$

7.4 Optimizaciones en el proceso de búsqueda

Al igual que hemos hecho con los algoritmos presentados en los temas anteriores, también aquí es posible realizar ciertas optimizaciones para reducir el coste computacional, y por tanto el tiempo de ejecución, de los algoritmos aquí planteados.

Una vez más, cabe matizar que hablaremos de optimizaciones en el sentido de mejorar la eficiencia de los algoritmos, aunque ello pueda, en la mayoría de los casos, ocasionar que el algoritmo pierda calidad en la traducción.

7.4.1 Reducción de la complejidad del algoritmo

Para reducir la complejidad del algoritmo intentaremos reducir el espacio de búsqueda, o lo que es lo mismo en este caso reducir el número de vecinos a explorar en cada iteración del algoritmo.

Como hemos visto en la sección anterior, las operaciones más costosas con diferencia son $swapSegments(\cdot)$, $swapSegmentsWithAlig(\cdot)$ $translateOneOrTwoWords(\cdot)$ y $translateAndInsertZFert(\cdot)$. En lo que sigue proponemos ciertas optimizaciones para reducir el número de veces que se deban ejecutar este tipo de operaciones, al igual que lo hemos hecho para los algoritmos presentados en capítulos anteriores.

A pesar de que los algoritmos voraces aquí expuestos son heurísticos por naturaleza, es decir no pueden garantizar la optimalidad de la solución, en gran número de casos serán capaces de escapar de máximos locales y por tanto alcanzar la solución óptima al proceso de búsqueda. Las optimizaciones que proponemos a continuación todas ellas llevarán asociados errores de búsqueda adicionales a los propios inherentes del algoritmo, aunque por otra parte permitirán reducir sustancialmente el coste computacional de los algoritmos. Al igual que en los casos anteriores estableceremos empíricamente los valores óptimos de los parámetros que definan las optimizaciones, siempre estableciendo un compromiso entre eficiencia y calidad en la traducción.

Reducción para la operación $swapSegments(\cdot)$

Lo aquí comentado también será válido para la operación $swapSegmentsWithAlig(\cdot)$. Estas operaciones están pensadas para dotar al algoritmo la capacidad de explorar traducciones entre frases que no mantengan la monotonicidad. Es decir, realizan todos los reordenamientos posibles entre las palabras de una hipótesis dada.

La optimización de estas operaciones consiste en limitar el tamaño máximo de los segmentos a intercambiar, es decir los segmentos podrán tener una longitud máxima S. Teniendo esto en cuenta el número de veces que se deberá ejecutar esta operación sera del orden de $O(S^4)$, con lo que el término de la complejidad asociado a esta operación pasará a ser $O(S^4 \cdot I)$. Esta simplificación de las operaciones podrá producir errores de búsqueda adicionales.

Reducción para la operación $translateOneOrTwoWords(\cdot)$

La posible optimización de esta operación pasa por, al igual que hemos hecho en los algoritmos de búsqueda anteriores, limitar el número de traducciones de una palabra f. Una vez más no resulta práctico considerar como posibles traducciones de una palabra f todas las palabras de $\mathcal{E}$. Por tanto limitaremos ese número a las W mejores traducciones inversas, y al conjunto de ellas le llamaremos $\mathcal{W}(f)$.

Con esta optimización el número de veces que deberá ejecutarse esta operación será proporcional a $O(J^2 \cdot W^2)$, con lo que el término de la complejidad asociado a esta operación pasará a ser $O(J^2 \cdot W^2 \cdot I^2)$.

Evidentemente si la traducción óptima de una palabra f (dentro de una frase en concreto), no se encuentra dentro de sus W mejores traducciones inversas entonces no se podrá alcanzar la traducción óptima. Es decir, esta limitación puede inducir errores de búsqueda.

Reducción para la operación $translateAndInsertZFert(\cdot)$

Para esta operación también es aplicable la optimización anterior. Además, y dado que la operación conlleva la inserción de una palabra de fertilidad 0, tampoco es práctico en este caso considerar todas las palabras del vocabulario $\mathcal{E}$ como palabras de fertilidad cero. Al igual que en caso de los algoritmos de pila, la optimización consiste en considerar un número máximo de palabras con gran probabilidad de tener fertilidad 0. Una vez más, llamaremos a este parámetro Z, y al conjunto de ellas $\mathcal{Z}$. Uniendo esta optimización a la anterior, el término de la complejidad asociado a esta operación pasa a ser $O(J \cdot W \cdot Z \cdot I^2)$.

Al igual que para la operación anterior, esta optimización podrá dar lugar a errores de búsqueda.

En el algoritmo 7.5 podemos ver como quedaría el algoritmo *ComputeNeighborhood(***a***)* aplicando las optimizaciones hasta ahora comen-

tadas. Sólo se incluyen los cambios con respecto al algoritmo no optimizado.

Nueva expresión de la complejidad con las optimizaciones S, W y Z

La expresión de la complejidad teniendo en cuenta estas optimizaciones pasará a ser:

$$O(S^4 \cdot I + J^2 \cdot W^2 \cdot I^2 + J \cdot W \cdot Z \cdot I^2 + I^2 \cdot I + I \cdot I) \equiv O(J^2 \cdot W^2 \cdot I^2)$$

Reducción de la complejidad de las operaciones

Para reducir aun más la complejidad del algoritmo realizaremos simplificaciones a algunas de las operaciones comentadas. La simplificación de las operaciones consistirá, en general, en no permitir que una operación realice más de una traducción, una inserción, un movimiento y/o un borrado de una palabra cada vez.

Como hemos comentado anteriormente restringiremos estas operaciones a realizar cambios en el alineamiento que afecten a lo sumo a una palabra, por lo tanto la operación $translateOneOrTwoWords(j, e, j', e')$ pasará a ser $translateOneWord(j, e)$, la operación $translateAndInsertZFert(j, e, ze)$ a $insertZFert(ze)$, y la operación $joinWords(i, i')$ a $attachWords(i, i')$. Además estas nuevas operaciones tendrán en cuenta las optimizaciones comentadas en la sección anterior.

Con las optimizaciones que se muestran a continuación y las comentadas en la sección anterior, el algoritmo 7.4 quedará como se muestra en el algoritmo 7.6.

A continuación comentaremos las nuevas operaciones:

translateOneWord(j, e)**:** Esta operación cambiará la traducción de la palabra origen, que ocupa la posición j, es decir e_{a_j}, a una nueva e. Se podrán dar las siguientes situaciones especiales en particular:

- Que e_{a_j} sea una palabra de fertilidad igual a 1, y que e sea la palabra vacía ($NULL$), entonces e_{a_j} se borrará de la hipótesis. Es decir, si la mejor traducción de la palabra origen pasa a ser la palabra vacía, entonces la palabra destino que hasta la fecha daba lugar a ella tendrá que desaparecer de la traducción.
- Que e_{a_j} sea $NULL$, entonces e se insertará en la posición que produzca el alineamiento de mayor probabilidad.
- Que $e_{a_j} = e$, en este caso la operación no tendrá efecto.

Esta operación, en el peor de los casos, tendrá un coste asintótico igual al de la operación $translateOneOrTwoWords(j, e, j', e')$, es decir $O(I^2)$, aunque deberá realizarse un total de $J \cdot W$ veces, teniendo en cuenta las optimizaciones anteriores.

Algoritmo: ***ComputeNeighborhood_opt_S+W+Z(a)***

entrada: $\mathbf{a}, \mathcal{W}(\cdot), \mathcal{Z}$
salida: $\mathcal{N}(\mathbf{a})$
  $\mathcal{N}(\mathbf{a}) = \emptyset$
  $//swapSegments$
  para $i = 1 \cdots I$ hacer
    $i' = i$
    mientras $i' < I$ y $i' < i + L$ hacer
      para $k = i' + 1 \cdots I$ hacer
        $k' = k$
        mientras $k' < I$ y $k' < k + L$ hacer
          $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{a}) \cup swapSegments(i, i', k, k')$
          $k' + +$
        fin_mientras
      fin_para
      $i' + +$
    fin_mientras
  fin_para
  $//translateOneOrTwoWords$
  para $j = 1 \cdots J$ y $j' = j + 1 \cdots J$ hacer
    para_todo $e \in \mathcal{W}(f_{a_j})$ y $e' \in \mathcal{W}(f_{a_{j'}})$ hacer
      $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{a}) \cup translateOneOrTwoWord(j, e, j', e')$
    fin_para
  fin_para
  $//translateAndInsertZFert$
  para $j = 1 \cdots J$ hacer
    para_todo $e \in \mathcal{W}(f_{a_j})$ y $ze \in \mathcal{Z}$ hacer
      $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{a}) \cup translateAndInsertZFert(j, e, ze)$
    fin_para
  fin_para
  $//joinWords$
  para $i = 1 \cdots I$ y $i' = i \cdots I$ hacer
    $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{a}) \cup joinWords(i, i') \cup joinWords(i', i)$
  fin_para
  $//removeZFert$
  para $i = 1 \cdots I$ hacer
    $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{a}) \cup removeZFert(i)$
  fin_para
  devolver: $\mathcal{N}(\mathbf{a})$

Algoritmo 7.5: Algoritmo para el cálculo del vecindario de un alineamiento dado con las optimizaciones para los parámetros W, Z y S.

$insertZFert(ze)$**:** Esta operación inserta la palabra ze en la posición que produzca el alineamiento de mayor probabilidad. Esta operación tendrá un coste asintótico idéntico al de la operación $translateAndInsertZFert(j, e, ze)$, es decir $O(I^2)$, pues el hecho de insertar la palabra de fertilidad cero requiere recorrer toda la hipótesis. En cambio el número de veces que esta operación deberá ejecutarse es Z, también teniendo en cuenta las optimizaciones anteriores.

$attachWords(i, i')$**:** Esta operación realizará la misma función que $joinWords(i, i')$ pero no se eliminará una de las palabras, sino que quedará como una palabra de fertilidad 0. El coste de esta operación se mantiene constante. El número de veces que se ejecutará será I^2.

Nueva expresión de la complejidad teniendo en cuenta la simplificación de las operaciones

A la vista de los costes parciales referentes a cada operación, y dado que todas ellas se ejecutan de forma secuencial, podemos decir que el coste asintótico del nuevo algoritmo *GreedySearchOpt* es del orden de:

$$O(S^4 \cdot I + J \cdot W \cdot I^2 + Z \cdot I^2 + I^2 \cdot I + I \cdot I) \equiv O(J \cdot W \cdot I^2)$$

7.5 Optimización del cálculo de la *puntuación*

Como hemos comentado en la sección 7.3 el coste del cálculo de la *puntuación* de una hipótesis, es decir, el cálculo de $Pr(\mathbf{e}) \cdot Pr_M(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$ es del orden de $O(I)$, el cual es un factor no despreciable dentro de lo que es la complejidad del algoritmo *GreedySearch*.

A la vista de algunos experimentos realizados se planteó llevar a cabo una optimización del cálculo de esta *puntuación*, dado que este coste podrá ser reducido considerablemente haciendo un estudio de la contribución a la *puntuación* de cada una de las operaciones definidas para el algoritmo voraz. En realidad, cada operación no es más que una modificación local a una hipótesis/alineamiento dado, con lo que sería interesante hacer una actualización con respecto a la operación aplicada en vez de recalcular, para cada aplicación de un operador, completamente la *puntuación*.

A título de ejemplo, a continuación veremos con detalle en que consiste la actualización de la *puntuación* con respecto a cada una de las operaciones definidas para el modelo 3, para posteriormente ver el resultado de su impacto en la eficiencia del algoritmo.

Cabe resaltar que las optimizaciones que se proponen a continuación en ningún momento se realizan de forma heurística, por lo tanto ningún error de búsqueda podrá ser atribuible a ellas.

Algoritmo: ***ComputeNeighborhoodOpt()***

entrada: $\mathbf{a}, \mathcal{W}(\cdot), \mathcal{Z}, L$
salida: $\mathcal{N}(\mathbf{a})$

```
N(a) = ∅
//swapSegments
para i = 1 ··· I hacer
  i' = i
  mientras i' < I y i' < i + L hacer
    para k = i' + 1 ··· I hacer
      k' = k
      mientras k' < I y k' < k + L hacer
        N(a) = N(a) ∪ swapSegments(i, i', k, k')
        N(a) = N(a) ∪ swapSegmentsWithAlig(i, i', k, k')
        k' + +
      fin_mientras
    fin_para
    i' + +
  fin_mientras
fin_para
//translateOneWord
para j = 1 ··· J hacer
  para_todo e ∈ W(f_{a_j}) hacer
    N(a) = N(a) ∪ translateOneWord(j, e)
  fin_para
fin_para
//insertZFert
para_todo ze ∈ Z hacer
  N(a) = N(a) ∪ insertZFert(ze)
fin_para
//removeZFert
para i = 1 ··· I hacer
  N(a) = N(a) ∪ removeZFert(i)
fin_para
//attachWords
para i = 1 ··· I y i' = i ··· I hacer
  N(a) = N(a) ∪ attachWords(i, i') ∪ attachWords(i', i)
fin_para
devolver: N(a)
```

Algoritmo 7.6: Algoritmo optimizado para el cálculo del vecindario de un alineamiento dado incluyendo las optimizaciones para los parámetros W, Z y S.

7.5.1 Actualización de la *puntuación* para el modelo 3

En esta sección veremos cómo actualizar la *puntuación* de una hipótesis con respecto a la probabilidad del modelo de traducción $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$. Concretamente lo veremos para el caso del modelo 3. Para el modelo 4, la actualización de la *puntuación* seguirá la misma filosofía que la aquí presentada, solo que en la mayoría de los casos los parámetros de distorsión se verán también afectados con lo que habrá que tenerlo en cuenta al realizar la actualización. Para diferenciar los algoritmos que incluyan estas optimizaciones de los no optimizados, sobre todo cuando hagamos alusión a ellos en la sección de experimentación, les añadiremos la etiqueta *, así por ejemplo el algoritmo *GreedySearchOpt* pasará a ser *GreedySearchOpt**.

Para realizar las actualizaciones de las *puntuaciones* vamos a suponer que además de conocer el alineamiento a_1^J, la hipótesis e_1^I, y la frase de entrada f_1^J, disponemos de la información de las palabras de la frase de entrada que son generadas por la hipótesis, es decir b_1^I, donde cada b_i, será:

$$b_i = \{j : a_j = i\}$$

Actualización de $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$ para la operación $swapSegments(i, i', k, k')$

Esta operación conlleva el intercambio de dos segmentos de palabras de la hipótesis, por lo tanto el alineamiento se mantendrá fijo. Esto es tanto como decir que la *puntuación* que se obtendría al aplicar esta operación solo afectaría al modelo léxico $t(\cdot)$, es decir no afectaría al modelo de distorsión $d(\cdot)$, ni al modelo de fertilidades $n(\cdot)$.

Definimos la probabilidad de un segmento cualquiera $e_i^{i'}$ como:

$$pseg(i, i') = \prod_{x=i}^{i'} \prod_{j \in b_x} t(f_j \mid e_x)$$

y la de un segmento que se verá desplazado o posiciones con respecto a su posición actual cómo:

$$pseg(i, i', o) = \prod_{x=i}^{i'} \prod_{j \in b_{x+o}} t(f_j \mid e_{x+o})$$

Dado que los segmentos podrán ser, a priori, de cualquier longitud habrá que matizar algunos casos en particular:

- $i' - i = k' - k$, es decir los segmentos tienen la misma longitud. En este caso solo se ven afectadas por el cambio las palabras que pertenecen a ambos segmentos, por tanto la actualización de la *puntuación* en este caso será:

$$Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})* = \frac{pseg(i, i', k - i)}{pseg(i, i')} \cdot \frac{pseg(k, k', -(k - i))}{pseg(k, k')}$$

- $i'-i \neq k'-k$, en este caso todas las palabras de la hipótesis en el rango $e_i^{k'}$ se ven afectadas por el intercambio. Por lo tanto en este caso la actualización de la *puntuación* deberá ser:

$$Pr(\mathbf{f},\mathbf{a}|\mathbf{e}) \quad * = \quad \frac{pseg(i,i',k'-i')}{pseg(i,i')} \cdot \frac{pseg(k,k',-(k-i))}{pseg(k,k')} \cdot \frac{pseg(i'+1,k-1,(k-k'+1)-(i-i'+1))}{pseg(i'+1,k-1)}$$

En el peor de los casos tendremos un coste para $Pr(\mathbf{f},\mathbf{a}|\mathbf{e})$ de $O(I)$ en caso de no restringir la longitud máxima de los segmentos, y de $O(S)$ si tenemos en cuenta dicha optimización.

Actualización de $Pr(\mathbf{f},\mathbf{a}|\mathbf{e})$ para la operación $swapSegmentsWithAlig(i,i',k,k')$

Para actualizar la *puntuación* en este caso basta con redefinir las probabilidades asociadas a un segmento. De esto modo definimos la probabilidad de un segmento cualquiera $e_i^{i'}$ como:

$$pseg(i,i') = \prod_{x=i}^{i'} \prod_{j \in b_x} t(f_j \mid e_x) \cdot d(x \mid j, J)$$

y la de un segmento que se verá desplazado o posiciones con respecto a su posición actual cómo:

$$pseg(i,i',o) = \prod_{x=i}^{i'} \prod_{j \in b_{x+o}} t(f_j \mid e_{x+o}) \cdot d(x+o \mid j, J) \ ,$$

dado que en este caso sí se verá afectada la *puntuación* con respecto a los parámetros de distorsión.

El coste de esta operación es idéntico a la anterior.

Actualización de $Pr(\mathbf{f},\mathbf{a}|\mathbf{e})$ para la operación $translateOneOrTwoWords(j,e,j',e')$

Esta operación tendrá un coste asintótico idéntico a la operación $translateOneWord(j,e)$, por tanto estudiaremos este último caso por tratar con solo una palabra en vez de con dos. Para esta operación habrá que distinguir entre los siguientes casos particulares:

- Que e_{a_j} sea una palabra de fertilidad 1, y que e sea la palabra vacía ($NULL$), entonces e_{a_j} se borrará de la hipótesis:

$$Pr(\mathbf{f},\mathbf{a}|\mathbf{e}) * = \frac{t(f_j \mid e_0)}{t(f_j \mid e_{a_j})} \cdot \frac{n(\phi_0+1 \mid e_0)}{n(\phi_0 \mid e_0)} \cdot \frac{1}{n(1 \mid e_{a_j})}$$

  En este caso el coste es actualizar $Pr(\mathbf{f},\mathbf{a}|\mathbf{e})$ es $O(cte)$.

- Que $a_j = 0$, entonces e se inserta en la posición que produzca el alineamiento de mayor probabilidad. La actualización de la *puntuación* en el caso de insertar la palabra en la posición i será:

$$Pr(\mathbf{f},\mathbf{a}|\mathbf{e})* = \frac{t(f_j|e)}{t(f_j|e_0)} \cdot \frac{n(\phi_0-1|e_0)}{n(\phi_0|e_0)} \cdot n(1 \mid e) \cdot d(i \mid j, J)$$
$$\text{para } (j' = 1; j' \leq J; j++)$$
$$\quad \text{si } (a_{j'} \geq i)\ Pr(\mathbf{f},\mathbf{a}|\mathbf{e})* = \frac{d(a_{j'}+1|j',J)}{d(a_{j'}|j',J)}$$

  Por tanto el coste de actualizar $Pr(\mathbf{f},\mathbf{a}|\mathbf{e})$ sera $O(J)$, y dado que esto habrá que repetirlo para toda posición en la cadena de salida, el coste de la operación en este caso será $O(I \cdot J)$.

- En el resto de casos solo habrá que actualizar la *puntuación* de acuerdo al traducción de la nueva palabra, es decir:

$$Pr(\mathbf{f},\mathbf{a}|\mathbf{e})* = \frac{t(f_j \mid e)}{t(f_j \mid e_{a_j})}$$

  donde otra vez el coste de la operación es $O(cte)$.

En definitiva, en el peor de los casos para esta operación tendremos un coste $O(I \cdot J)$.

Actualización de $Pr(\mathbf{f},\mathbf{a}|\mathbf{e})$ para la operación $translateAndInsertZFert(j,e,ze)$

En este caso existe la misma casuística que en la operación anterior, con la diferencia que además habrá que insertar una palabra de fertilidad 0.

La actualización de la *puntuación* en el caso de insertar la palabra en la posición i será:

$$Pr(\mathbf{f},\mathbf{a}|\mathbf{e})* = n(0 \mid ze)$$
$$\text{para } (j' = 1; j' \leq J; i++)$$
$$\quad \text{si } (a_{j'} \geq i)\ Pr(\mathbf{f},\mathbf{a}|\mathbf{e})* = \frac{d(a_{j'}+1|j',J)}{d(a_{j'}|j',J)}$$

Al igual que en el caso anterior, dado que esto habrá que repetirlo para toda posición en la cadena de salida, el coste de esta operación en este caso será $O(I \cdot J)$, que será igual al de la operación $insertZFert(j,e,ze)$.

Actualización de $Pr(\mathbf{f},\mathbf{a}|\mathbf{e})$ para la operación $removeZFert(i)$

La actualización de la *puntuación* en el caso de eliminar la palabra de la posición i será:

$$Pr(\mathbf{f},\mathbf{a}|\mathbf{e})* = \frac{1}{n(0|e_i)}$$

$$\text{para } (j' = 1; j' \leq J; i++)$$

$$\quad \text{si } (a_{j'} \geq i)\ Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})* = \frac{d(a_{j'}-1|j',J)}{d(a_{j'}|j',J)}$$

Por tanto, el coste de esta operación será $O(I \cdot J)$.

Actualización de $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$ para la operación $joinWords(i, i')$

Supongamos que las palabras alineadas con $e_{i'}$ se unen a e_i, y por lo tanto $e_{i'}$ será eliminada de la hipótesis, entonces la *puntuación* se actualizará del siguiente modo:

$$Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})* = \frac{n(\phi_i+\phi_{i'}|e_i)}{n(\phi_i|e_i)\cdot n(\phi_{i'}|e_{i'})} \cdot \prod_{j \in b_{i'}} \frac{t(f_j|e_i)}{t(f_j|e_{i'})\cdot d(i'|j,J)}$$

$$\text{para } (j' = 1; j' \leq J; i++)$$

$$\quad \text{si } (a_{j'} > i') \text{ entonces } Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})* = \frac{d(a_{j'}-1|j',J)}{d(a_{j'}|j',J)}$$

En este caso el coste de la operación será $O(I \cdot J)$.

En cambio la actualización de la *puntuación* para la operación $attachWords(i, i')$ será:

$$Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})* = \frac{n(\phi_i + \phi_{i'} \mid e_i) \cdot n(0 \mid e_{i'})}{n(\phi_i \mid e_i) \cdot n(\phi_{i'} \mid e_{i'})} \cdot \prod_{j \in b_{i'}} \frac{t(f_j \mid e_i)}{t(f_j \mid e_{i'}) \cdot d(i' \mid j, J)} \ ,$$

es decir de coste $O(cte)$, por tanto el coste de la operación será $O(I)$.

7.5.2 Actualización de la *puntuación* para el modelo de lenguaje

Como ya vimos en su momento, el coste de calcular $Pr(\mathbf{e})$ es del orden de $O(I)$, aunque este coste se puede simplificar aplicando actualizaciones similares a las expuestas para el modelo de traducción, es decir actualizando el valor de $Pr(\mathbf{e})$ solamente para los valores $p(e_i \mid h(i))$ que se ven afectados por la operación. Estas actualizaciones, en media, reducirán la complejidad del cálculo de $Pr(\mathbf{e})$, pero en el peor de los casos (operación $swapSegments(\cdot)$) sera también del orden de $O(I)$.

Los detalles de cómo actualizar el valor de $Pr(\mathbf{e})$ utilizando n-gramas y para cada operación en concreto carecen de interés dada la trivialidad de los mismos. De cualquier forma esta optimización también se ha llevado a cabo en la implementación de los algoritmos.

7.5.3 Expresión final de la complejidad

Teniendo en cuenta las optimizaciones presentadas en la actualización de la *puntuación*, la complejidad del algoritmo *GreedySearch* para el modelo 3 queda del siguiente modo:

$$O(J^{2\cdot} \mid \mathcal{E} \mid^2 \cdot I \cdot J) \equiv O(J^{3\cdot} \mid \mathcal{E} \mid^2 \cdot I)$$

Teniendo en cuenta las optimizaciones relacionadas con la reducción del espacio de búsqueda tendríamos:

$$O(J^2 \cdot W^2 \cdot I \cdot J) \equiv O(J^3 \cdot W^2 \cdot I)$$

Y teniendo en cuenta además las simplificaciones introducidas a las operaciones tendríamos finalmente:

$$O(J \cdot W \cdot I \cdot J) \equiv O(J^2 \cdot W \cdot I)$$

En realidad a la vista de estas expresiones de la complejidad teórica del algoritmo parece que no se gana nada teniendo en cuenta las optimizaciones propuestas para el cálculo de $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$, pues en realidad la complejidad teórica de este cálculo ha pasado de $O(I)$ a $O(J)$.

En cambio, como podremos ver en la sección de experimentación (ver sección 7.6.2) la reducción real en tiempo de ejecución es tremenda pues según hemos comentado la complejidad teórica se ha calculado en el peor de los casos (casos particulares de las operaciones $translateOneOrTwoWords(\cdot)$ y $translateAndInsertZFert(\cdot)$), y en media el coste de actualizar el cálculo de la *puntuación* es prácticamente constante, con lo que podríamos decir que el coste final de estos algoritmos es de $\Theta(J^2 \cdot W^2 \cdot I)$ o de $\Theta(J \cdot W \cdot I)$ si utilizamos las operaciones simplificadas. Es decir, con la optimización del cálculo de la *puntuación* hemos reducido la complejidad del algoritmo en un orden de magnitud con respecto a la talla del problema, es decir J, la longitud de la cadena de entrada.

7.6 Discusión de los algoritmos voraces

En este sección vamos a discutir las cualidades de los algoritmos de voraces experimentalmente. Para ello llevaremos a cabo una serie de experimentos utilizando el mismo corpus utilizado para los algoritmos de búsqueda presentados en los capítulos anteriores (ver sección 5.8, tabla 5.1, página 127), para que la comparación entre ellas se realice de forma objetiva. Del mismo modo el estudio se realizará siguiendo los mismos pasos, en concreto:

1. Establecer un algoritmo sobre el que se realizarán las pruebas.

2. Hacer un recorrido por las distintas optimizaciones planteadas, experimentando sobre su efecto en la eficiencia y la tasa de aciertos del algoritmo. En concreto se harán diversas pruebas con los valores de los parámetros W, Z y S.

3. En tercer lugar haremos una comparativa sobre la influencia que tienen en la eficiencia, el algoritmo que se escoja, el modelo de traducción, y el tamaño de frase.

Al igual que en los dos capítulos anteriores, los modelos de traducción se han entrenado con el conjunto de entrenamiento mostrado en la tabla 2.2, es decir un corpus de 10,000 pares de frases, utilizando el esquema de entrenamiento $1^5 2^5 3^5 4^5 5^5$. Con respecto al modelo de lenguaje, una vez más, se han utilizado trigramas con la técnica de descuento *GoodTuring* y ha sido entrenado con las correspondientes 10,000 frases para el lenguaje de salida. Del mismo modo, como resultado de este estudio, escogeremos un traductor óptimo con el que obtendremos más resultados en la sección siguiente.

Una vez más, en las tablas relativas a los experimentos que expondremos a continuación se incluye:

- El tiempo medio de ejecución en segundos (**secs**) por frase.
- El modelo de error, es decir: porcentaje errores de búsqueda (**ErrBusq**), porcentaje errores del modelo (**ErrModel**) y porcentaje de aciertos (**Aciertos**).
- Las tasas de error (WER y PER), en tanto por ciento de error.

7.6.1 Establecimiento de los parámetros de los algoritmos voraces

Vamos a fijar por defecto (y en base a experimentaciones preliminares) las características básicas del traductor de la siguiente manera:

- Algoritmo elegido: *GreedySearch* con *ComputeNeighborhood_opt_S+W+Z*
- Modelo de traducción utilizado: modelo 3
- Parámetros: $W = 12$, $Z = 24$, $S = 9$
- Inicialización: Viterbi

Influencia del número de traducciones inversas (parámetro W)

En la tabla 7.1 podemos ver la influencia del parámetro W con respecto a la eficiencia y aciertos en la traducción. A la vista de estos resultados parece claro que este parámetro no es determinante, al igual que sí lo era para los

demás algoritmos expuestos, en cuanto a eficiencia y la calidad en la traducción. También podemos ver que a partir de un valor de 20 no se obtiene prácticamente mejora alguna en la calidad, y sí una ligera depreciación en la eficiencia, por tanto establecer un valor de $W = 20$ sería suficiente.

W	secs	ErrBusq	ErrModel	Aciertos	WER	PER
5	3.00	53.85	21.40	24.75	22.23	18.20
10	4.30	50.84	23.75	25.42	21.67	17.40
15	5.18	50.84	23.75	25.42	21.74	17.44
20	5.83	50.84	23.75	25.42	21.74	17.44
25	6.13	50.84	23.75	25.42	21.71	17.40
30	6.41	50.84	23.75	25.42	21.71	17.40
35	6.63	50.84	23.75	25.42	21.67	17.40
40	6.79	50.84	23.75	25.42	21.67	17.40
45	6.91	51.17	23.75	25.08	21.78	17.51
50	7.09	50.50	24.08	25.42	21.67	17.40

Tabla 7.1: Influencia del número de traducciones inversas para una palabra origen dada (parámetro W).

Influencia del número de palabras de fertilidad cero (parámetro Z)

El parámetro Z establece el número máximo de traducciones de palabras de fertilidad cero que se consideran al ejecutar el bucle de operaciones $translateAndInsertZFert(\cdot)$, es por tanto un parámetro vital del algoritmo debido a su influencia en la calidad de la traducción y en la eficiencia. En la tabla 7.2 se muestran pruebas con distintos valores de este parámetro. Como puede apreciarse, este parámetro tiene un impacto mucho mayor en la eficiencia y en los aciertos que W. Por ejemplo, podemos ver que al pasar de un valor de 5 a un valor de 30, los errores de búsqueda se ven reducidos en un 50%. Por contra, el tiempo medio por frase se ve aumentado en un factor aproximadamente igual a 5. Siguiendo con el criterio de establecer un compromiso entre eficiencia y eficacia, parece razonable establecer este parámetro al valor de $Z = 30$, pues a partir de ahí no se obtienen mejoras sustanciales en cuanto a calidad y sí depreciaciones importantes en el tiempo de ejecución.

A la vista de estos resultados y los anteriores (parámetro W) sería interesante hacer un estudio bidimensional del efecto de estos dos parámetros para ver si realmente el parámetro W afecta o no a las prestaciones de estos algoritmos.

Influencia del tamaño de los segmentos a intercambiar (parámetro S)

El parámetro S establece el número tamaño máximo de segmentos a intercambiar por tanto afecta al bucle de operaciones del tipo $swapSegments(\cdot)$. En

Z	secs	ErrBusq	ErrModel	Aciertos	WER	PER
5	1.22	71.24	15.72	13.04	25.36	21.12
10	2.10	61.54	20.74	17.73	23.62	19.07
15	2.99	57.86	22.74	19.40	22.89	18.30
20	3.90	56.52	22.74	20.74	22.65	18.10
25	4.87	51.84	23.41	24.75	22.06	17.85
30	5.73	48.83	25.08	26.09	21.78	17.65
35	6.62	48.83	25.08	26.09	21.78	17.65
40	7.47	48.83	25.08	26.09	21.78	17.65
45	8.34	48.83	25.08	26.09	21.60	17.68
50	9.24	48.83	25.08	26.09	21.60	17.68

Tabla 7.2: Influencia del número de palabras de fertilidad cero consideradas (parámetro Z).

la tabla 7.3 se muestran los resultados para distintos valores de este parámetro. A la vista de ellos podemos ver que no existe prácticamente efecto de este parámetro en cuanto a la eficiencia, en cambio si afecta a la calidad de la traducción, pues al pasar de un valor de 1 a un valor de 6, los errores de búsqueda se reducen en aproximadamente un 20%. Este mismo efecto se puede ver en los índices de error, obteniéndose una mejora de más de 8 puntos.

S	secs	ErrBusq	ErrModel	Aciertos	WER	PER
1	4.51	59.87	18.39	21.74	29.91	20.35
2	4.57	58.19	20.74	21.07	29.39	19.76
3	4.61	56.86	20.07	23.08	27.86	19.38
4	4.66	55.85	20.07	24.08	25.84	18.83
5	4.64	51.84	23.41	24.75	22.82	17.96
6	4.69	51.51	23.75	24.75	21.88	17.58
7	4.70	51.51	23.75	24.75	21.88	17.58
8	4.69	51.51	23.75	24.75	21.88	17.58
9	4.70	51.51	23.75	24.75	21.88	17.58

Tabla 7.3: Influencia del tamaño máximo de los segmentos a intercambiar en la operación $swapSegments(\cdot)$ (parámetro S).

Como vemos, este parámetro tiene un impacto relevante en la eficiencia del algoritmo pues las operaciones del tipo $swapSegments(\cdot)$ son costosas y se ejecutan gran cantidad de veces.

7.6.2 Estudio empírico de eficiencia

Vamos a continuar nuestro estudio usando un traductor con las mismas características básicas que se usaron para el establecimiento de los los parámetros.

Determinación de la influencia del algoritmo

En la tabla 7.4 podemos ver los resultados de la influencia del algoritmo utilizado. En esta tabla $GSoSWZ$ hace referencia al algoritmo *GreedySearch* con *ComputeNeighborhood_opt_S+W+Z* y $GSOpt$ al mismo con *ComputeNeighborhoodOpt*, y los * a sus versiones optimizadas en el cálculo de la *puntuación*. En este experimento hemos utilizado el modelo 3 (de ahí que se obtengan peores resultados), para poder ver el efecto de la optimización con respecto al cálculo de la *puntuación*.

A la vista de estos resultados podemos ver la depreciación del algoritmo, en cuanto a calidad en la traducción se refiere, con respecto a las optimizaciones de las operaciones, aunque ello suponga una sustancial ganancia en eficiencia. En cambio la mejora en la eficiencia que incluye la optimización del cálculo de la *puntuación* es tremenda, de modo que la ganancia que podríamos obtener con la optimización de las operaciones se ve en gran medida subsanada con dicha optimización. Por tanto, ya podemos adelantar que el estudio e implementación de la optimización del cálculo de la *puntuación* para el resto de los modelos se hace imprescindible para obtener tiempos de ejecución realmente competitivos.

Alg.	**secs**	**ErrBus**	**ErrModel**	**Aciertos**	**WER**	**PER**
$GSoSWZ^*$	0.52	51.51	23.75	24.75	21.88	17.58
$GSoSWZ$	3.66	51.51	23.75	24.75	21.88	17.58
$GSOpt^*$	0.08	71.24	13.71	15.05	26.26	21.33
$GSOpt$	0.39	71.24	13.71	15.05	26.26	21.33

Tabla 7.4: Influencia del algoritmo.

Como era de esperar los resultados obtenidos con la optimización del cálculo de la *puntuación* no introducen errores de búsqueda adicionales.

Estudio empírico del coste de las operaciones

A la vista de los resultados de la influencia del algoritmo parece interesante realizar un estudio del coste medio de las distintas operaciones involucradas en cada algoritmo. En la tabla 7.5 podemos ver el tiempo medio (en segundos) utilizado en las distintas operaciones para las distintas versiones del algoritmo *GreedySearch*.

El tiempo medio de las operaciones $translateAndInsertZFert$ y $translateOneOrTwoWords$ se da conjuntamente, pues en la implementa-

Operación	GSoWZ	GSoWZ*	GSOpt	GSOpt*
$swapSegments$	0.050	0.015	0.052	0.016
$removeZFertWords$	0.000	0.000	0.000	0.000
$joinWords$	0.005	0.000	–	–
$attachWords$	–	–	0.005	0.000
$translateAndInsertZfert+$ $translateOneOrTwoWords$	0.799	0.113	–	–
$insertZfert+$ $translateOneWord$	–	–	0.023	0.003

Tabla 7.5: Tiempo medio (en segundos) por operación para distintas versiones del algoritmo voraz.

ción se ha hecho uso de la parte común de ambas para agilizar su ejecución. Por tanto no resulta sencillo separar el tiempo medio utilizado para ellas pues ello supondría cambiar la implementación y con ello elevar el tiempo medio de ejecución del algoritmo. De cualquier forma podemos ver que dichas operaciones son las más costosas con diferencia, hecho que verifica el estudio de complejidad realizado. Según los tiempos mostrados en la tabla encontramos la justificación de la mejora obtenida cuando se realiza la optimización del cálculo de la *puntuación*, pues es precisamente en las operaciones más costosas donde mayor relevancia tiene dicha optimización.

Como podemos ver los tiempos asociados a las operaciones $removeZFertWords$, $joinWords$, y $attachWords$, son prácticamente despreciables, de ahí los valores de 0.000, pues en la mayoría de los casos el tiempo invertido en esas operaciones no alcanza la milésima de segundo.

Estos resultados corroboran los expuestos en la tabla 7.4 anterior pues se puede ver que la reducción de las operaciones $translateAndInsertZfert(\cdot)$ y $translateOneOrTwoWords(\cdot)$ son las que marcan la diferencia entre los dos algoritmos.

Determinación de la influencia del tamaño de la frase

Al igual que hicimos con los algoritmos de pila, parece interesante realizar un experimento que refleje la influencia del tamaño de la frase en las prestaciones de los algoritmos.

Para este experimento hemos elegido distintos conjuntos de test (de la tarea EuTrans-I), de distintas longitudes, concretamente hemos elegido 100 frases de test longitudes 4, 6, 8, 10, 12, 15 y entre 17 y 20 palabras. En la tabla 7.6 podemos ver la información del modelo de error y tiempo medio de traducción de las frases, donde se hace patente una influencia muy estrecha entre la longitud de frase y el coste temporal del algoritmo, pues el tiempo de ejecución crece exponencialmente con el tamaño de la frase. En este hecho también influye el

número de medio de iteraciones que el algoritmo debe de realizar hasta converger, pues como era de esperar a mayor longitud mayor número de iteraciones se deben realizar hasta alcanzar la convergencia.

En esta misma tabla también podemos apreciar cómo el número de errores de búsqueda aumenta con la longitud de la frase, cosa que también era de esperar pues a mayor espacio de búsqueda más fácil será alcanzar óptimos locales. En cambio, a la vista de las tasas de error para este experimento, podemos ver que esa diferencia no parece estar tan clara en lo que a la tasa de aciertos se refiere.

Long.	Iter.	secs	ErrBusq	ErrModel	Aciertos	WER	PER
4	2	0.07	20	40	40	38.27	33.33
6	2	0.55	35	35	30	24.86	20.90
8	4	2.35	49	30	21	21.69	18.84
10	4	5.45	56	23	21	19.34	15.57
12	5	13.38	64	18	18	23.78	18.24
15	7	64.79	84	10	6	27.68	17.46
17-20	8	97.55	87	9	4	29.32	17.46
media	4.57	26.30	48.43	22.14	27.54	26.42	20.26

Tabla 7.6: Influencia de la longitud de frase en el modelo y tasas de error.

Del mismo modo que hemos hecho en el apartado anterior podemos ver que tiempo se dedica a cada operación dependiendo de la longitud de la frase de entrada. En la tabla 7.7 podemos ver dicha relación y como era de esperar la influencia de la longitud de la frase en el tiempo de las operaciones más costosas es tremendo.

Operación	t4	t6	t8	10	12	15	17-20
$swapSegments$	0.0	0.004	0.038	0.104	0.203	1.11	1.781
$removZFertWords$	0.0	0.000	0.000	0.000			
$joinWords$	0.0	0.000	0.007	0.005	0.013	0.031	0.039
$trAndInsZfert+$ $trOneOrTwoWords$	0.033	0.194	0.514	1.174	2.234	7.102	9.765

Tabla 7.7: Tiempo medio (en segundos) por operación para distintos tamaños del corpus de test.

Determinación de la influencia del modelo utilizado

En la tabla 7.8 podemos ver los resultados de la influencia del modelo utilizado. En ella se observa que los mejores resultados se obtienen para el modelo 4, aunque sorprendentemente son prácticamente idénticos a los obtenidos con el

modelo 2. Es más, el tiempo medio por frase para el modelo 4 es sustancialmente mayor, como era de esperar.

M	secs	ErrBus	ErrModel	Aciertos	WER	PER
1	1.18	99.00	0.00	1.00	40.12	28.48
2	3.91	48.49	22.74	28.76	20.39	16.22
H	10.26	49.16	24.08	26.76	21.60	16.92
3	4.67	51.51	23.75	24.75	21.88	17.58
4	13.76	46.82	26.09	27.09	20.22	17.78
5	13.84	56.86	21.07	22.07	21.40	17.96

Tabla 7.8: Influencia del modelo.

7.7 Resultados de traducción

Vamos a llevar a cabo una serie de experimentos utilizando un traductor con los parámetros y características establecidos en la sección anterior. Para hacer dichos experimentos se han escogido dos tareas distintas: EuTrans-I, y Hansards, cuyas características fueron descritas en el capítulo 2, sección 2.7.

7.7.1 Elección del algoritmo

Vistos los resultados obtenidos en la experimentación llevada a cabo en esta sección, e intentando establecer un compromiso entre eficiencia y aciertos en la traducción, podemos concluir que una buena elección de un traductor será aquel que: utilice el algoritmo $GSoSWZ^*$, con inicialización por Viterbi. Como ya hemos comentado, será importante escoger un buen valor para los parámetros de optimización, concretamente hemos escogido los valores de $W = 20$, $Z = 25$, $S = 10$, y el modelo 3. De todas formas también daremos resultados con el modelo 4 pues es el que obtiene los mejores resultados con diferencia.

7.7.2 Resultados con la tarea EuTrans-I

En la tabla 7.9 se muestran los índices relativos a la traducción para el corpus de longitudes menor o igual que 15, de la tarea EuTrans-I. En la primera fila de la tabla se muestran los resultados obtenidos utilizando el modelo 3 y en la segunda los resultados utilizando el modelo 4.

secs	ErrBus	ErrModel	Aciertos	WER	PER
18.71	61.27	20.49	18.25	24.81	18.62
165.99	53.04	23.26	23.70	20.02	16.24

Tabla 7.9: Resultados con la tarea EuTrans-I, para frases de longitud menor e igual que 15, para los algoritmos voraces.

7.7.3 Resultados con la tarea HANSARDS

Para la tarea de HANSARDS hemos realizado experimentos para cinco corpus de 100 frases cada uno, de frases de longitud 4, 6, 8, 10 y 12, dada la complejidad que esta tarea conlleva.

Para tamaños mayores el tiempo de computación es enorme, debido principalmente a la operaciones $translateAndInsertZfert(\cdot)$ y $translateOneOrTwoWords(\cdot)$ como ya hemos comentado en la sección anterior.

Para esta tareas, y de acuerdo a una experimentación previa (ver apéndice C), hemos fijado el valor de los parámetros $W = 12$, $Z = 30$, y $S = 10$.

Los modelos de traducción y lenguaje utilizados en este experimento han sido los mismos que los que se en los dos capítulos anteriores (ver sección 5.9.2, página 134).

En la tabla 7.10 podemos ver la información del modelo de error y tiempo medio de traducción de las frases, así como las tasas de error para este experimento. Los resultados se presentan para los distintos corpus utilizados, de modo que también se puede ver la influencia de la longitud de la frase en el comportamiento del algoritmo.

Long.	It.	segs	ErrBusq	ErrModel	Aciertos	WER	PER
4	1	0.45	35.00	33.00	32.00	43.89	43.39
6	2	2.70	10.00	80.00	10.00	52.45	49.24
8	3	8.54	18.00	78.00	4.00	58.11	52.79
10	4	22.78	10.00	87.00	3.00	61.40	55.34
12	5	50.85	2.00	97.00	1.00	64.01	54.41
media	3	17.06	15.00	75.00	10.00	55.97	51.03

Tabla 7.10: Influencia de la longitud de la frase en el modelo y tasas de error para la tarea HANSARDS, para los algoritmos voraces.

7.8 Conclusiones

En este tema hemos expuesto, implementado y estudiado una familia de algoritmos voraces inspirados en el algoritmo voraz propuesto en [Germann et al. 01] para el modelo 4.

De estos algoritmos hemos estudiado su complejidad teórica, y diversas formas de optimizar los algoritmos, realizando un estudio empírico de eficiencia para los distintos parámetros de optimización y las distintas versiones de los algoritmos.

Por otra parte hemos ampliado el algoritmo propuesto en [Germann et al. 01] para todos los modelos de IBM y el modelo HMM. Es más, los algoritmos voraces aquí expuestos son susceptibles de ser usados con cualquier modelo de traducción que permita calcular la probabilidad de un alineamiento dado entre un par de frases origen-destino. De hecho se han implementado pensando en ello, con lo que para añadir un nuevo modelo de traducción dentro de estos algoritmos bastaría con insertar la función que calcule la probabilidad $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$ para dicho modelo.

De acuerdo con la experimentación llevada a cabo y a la vista de los resultados de traducción obtenidos en la sección anterior, podemos concluir que:

- Los algoritmos voraces son susceptibles de ser utilizados en tareas de complejidad reducida, aunque a tenor de los resultados obtenidos, las tasas de acierto para la tarea HANSARDS son bastantes parecidas a las obtenidas con el resto de algoritmos de búsqueda.

- Estableciendo un compromiso entre eficiencia y calidad, se pueden obtener traducciones aceptables en unos tiempos de ejecución medios, siendo la longitud de la cadena (por su parte tamaño del problema en la complejidad teórica) el factor determinante en su eficiencia.

- Dada la naturaleza heurística de estos algoritmos no parece razonable que sean utilizados en sistemas de traducción que requieran de una calidad alta, en cambio, sí en sistemas de ayuda a la traducción, en los que la calidad puede que no sea un parámetro tan relevante cómo la velocidad de traducción.

- La optimización de la reducción de la complejidad de las operaciones arroja una gran mejoría en cuanto a eficiencia pero también degrada sustancialmente las prestaciones del algoritmo.

- En cambio, la optimización no heurística del cálculo de la *puntuación* obtiene unas mejoras tremendas en cuanto a eficiencia, sin que ello merme las prestaciones del algoritmo.

Por último destacar una gran ventaja que ofrecen estos algoritmos, y es que la inclusión de los modelos basados en máxima entropía es extremadamente senci-

lla, lo que a nuestro a juicio los hace bastante atractivos para seguir investigando en ellos en el futuro.

IV

Conclusiones y bibliografía

Capítulo 8

Conclusiones

8.1 Resumen resultados

Esta sección la hemos dedicado para exponer de forma resumida los resultados experimentales más relevantes llevados a cabo en el desarrollo de la tesis.

Por un lado haremos un resumen de los resultados obtenidos en la parte de modelado y por otro los obtenidos en la parte de algoritmos de búsqueda.

8.1.1 Resumen de resultados de calidad en los alineamientos

A continuación se presentan los mejores resultados obtenidos con la integración de los modelos de traducción basados en máxima entropía en el entrenamiento de los modelos de IBM, en comparación con los resultados obtenidos con los modelos convencionales. En la tabla 8.1, se presentan los resultados para las tareas Verbmobil y Hansards para el esquema de entrenamiento completo, es decir $1^5 2^5 3^3 4^3 5^3$, y los distintos tamaños del corpus de entrenamiento utilizados.

A la vista de los resultados podemos ver que el ratio de errores de alineamiento mejora cuando utilizamos los modelos léxicos dependientes del contexto. Para la tarea Verbmobil la mejora es menor que para la tarea Hansards, lo cual se debe al hecho de que el resultado base de calidad en el alineamiento es bastante bueno.

Como era de esperar, el entrenamiento por ME tiene mayor importancia cuanto mayor es el tamaño del corpus a utilizar. Para los corpus más pequeños, el número de eventos de entrenamiento para los modelos por ME es bastante reducido, hecho que provoca que en ciertas ocasiones no se puedan desambiguar algunas traducciones/alineamientos para diferentes contextos.

Corpus	Modelo	Tamaño del corpus		
		0.5K	8K	128K
HANSARDS	IBM	41.5	24.8	16.2
	IBM+ME	41.2	24.3	14.3
VERBMOBIL	IBM	22.6	9.9	7.2
	IBM+ME	22.3	9.6	6.8

Tabla 8.1: Resumen de AER [%] para las tareas HANSARDS y VERBMOBIL para distintos tamaños de corpus de entrenamiento.

8.1.2 Resumen de resultados de traducción

Esta sección la hemos dedicado para exponer de forma resumida los resultados de traducción llevados a cabo en la parte de algoritmos de búsqueda. Del mismo modo aprovecharemos para realizar una comparativa entre ellos, y otros resultados obtenidos por otros autores, y destacar las características más relevantes de cada aproximación, distinguiendo entre las dos tareas que hemos utilizado en todo el desarrollo, EUTRANS-I y HANSARDS.

Resumen de resultados para la tarea EUTRANS-I

En la tabla 8.2 podemos ver un resumen de los resultados de traducción para esta tarea siguiendo las distintas aproximaciones expuestas en la tesis. En ella se muestran el tiempo medio de traducción en segundos, y el modelo de error. La entrada **PD** se refiere al algoritmo por programación dinámica *IterativeSearchM2* y la entrada **PD-V4** se refiere al algoritmo por programación dinámica *IterativeSearchVitM4*.

Aprox.	**segs**	**ErrBus**	**ErrModel**	**Aciertos**
PD	55.70	5.50	55.24	39.26
PD-V4	69.46	12.18	45.11	42.72
Pila	87.12	18.44	44.08	37.48
Voraz	18.71	61.27	20.49	18.25

Tabla 8.2: Resumen de resultados para la tarea EUTRANS-I. Tiempo de traducción (en segundos) y modelo de error (% de errores de búsqueda, del modelo, y aciertos a nivel de frase) para distintos algoritmos de búsqueda.

En la tabla 8.3 se muestran las tasas de error obtenidas con las distintas aproximaciones así como los resultados presentados en [Och 02] para esta misma tarea. En dicha tabla la entrada **SWB** corresponde al algoritmo *SingleWord-Based* basado en el modelo HMM presentado en [Tillmann 01], y la entrada **AT**

corresponde al algoritmo *AlignmentTemplates* basado en modelos de traducción de grupos de palabras y plantillas presentado en [Och 02].

Aprox.	WER	PER
PD	12.71	10.50
PD-V4	10.19	9.44
Pila	14.21	11.13
Voraz	24.81	18.62
SWB	10.80	10.00
AT	4.40	2.90

Tabla 8.3: Resumen y comparación de resultados de traducción (en % de error a nivel de palabra) siguiendo distintas aproximaciones para la tarea EuTrans-I.

A la vista de los resultados de la tabla 8.2 podemos decir que:

- Los algoritmos basados en programación dinámica obtienen los mejores resultados, con diferencia, frente a las otras dos aproximaciones.
- Los algoritmos voraces, son de largo los más rápidos pero por contra arrojan los peores resultados, obteniéndose gran cantidad de errores de búsqueda.
- Los algoritmos de pila establecen un término medio en cuanto a calidad y eficiencia. El hecho de que estos algoritmos arrojen resultados peores a los de programación dinámica radica en las restricciones que debemos imponerles para obtener unos tiempos de respuesta aceptables.

A la vista de los resultados de la tabla 8.3 podemos ver que la mejor aproximación con diferencia es la **AT**, hecho justificado por la propia filosofía de la aproximación. Los resultados obtenidos con la aproximación **SWB** son comparables con los presentados en esta tesis. De cualquier forma cabe matizar que los resultados obtenidos con estas dos aproximaciones realizan importantes preprocesos a los corpus de entrenamiento y test para obtener mejores modelos de traducción y en general mejores resultados de traducción. Para más detalles acerca de los preprocesos realizados nos remitimos a los documentos en los que se presentan dichas aproximaciones. Por otra parte también destacar que todos los resultados de traducción expuestos en esta tesis se han obtenido sin realizar ningún tipo de preproceso a los corpus de entrenamiento y test.

Resumen de resultados para la tarea Hansards

En la tabla 8.5 se muestran las tasas de error obtenidas con las distintas aproximaciones así como los resultados presentados en [Och 02] para esta misma tarea.

Algoritmo	secs	ErrBus	ErrModel	Aciertos
PD	102.87	3.00	81.20	16.20
Pila	163.13	12.00	78.60	9.40
Voraz	17.06	15.00	75.00	10.00

Tabla 8.4: Resumen de resultados para la tarea HANSARDS. Tiempo de traducción (en segundos) y modelo de error (% de errores de búsqueda, del modelo, y aciertos a nivel de frase) para distintos algoritmos de búsqueda.

Aprox.	WER	PER
PD	50.53	46.78
Pila	54.16	51.30
Voraz	55.97	51.03
SWB	65.50	53.00
SWB+IBM	64.90	51.40
AT	61.50	49.20

Tabla 8.5: Resumen y comparación de resultados de traducción (en % de error a nivel de palabra) siguiendo distintas aproximaciones para la tarea HANSARDS.

A la vista de los resultados de la tabla 8.4 podemos sacar las mismas conclusiones que para la tarea EUTRANS-I, obteniéndose los mejores resultados una vez más con la aproximación por programación dinámica, donde cabe destacar el bajísimo número de errores de búsqueda que comete esta aproximación. En cuanto a los algoritmos de pila y los voraces podemos concluir que se comportan de forma muy parecida aunque una vez más el tiempo de respuesta de los algoritmos voraces supera con creces a los algoritmos de pila.

Los resultados de la tabla 8.5, para las aproximaciones **AT**, **SWB** y **SWB+IBM** (SWB utilizando reordenamientos al estilo IBM) no son directamente comparables con el resto. Esto es debido a que los resultados obtenidos con dichas aproximaciones se utilizaron los corpus de entrenamiento y test completos, a saber 1470473 y 5432 frases respectivamente. De cualquier forma los resultados expuestos en la tabla nos dan una idea de cómo se comportan las aproximaciones aquí expuestas con respecto a otras, y podemos decir que los resultados obtenidos con nuestras aproximaciones no difieren en general con otras. De todos modos las tasas de error para esta tarea son relativamente altas en general para cualquier aproximación, con lo que podemos concluir que aún estamos lejos de obtener buenos resultados de traducción para tareas de complejidad elevada.

8.2 Resumen de aportaciones

Con respecto a los objetivos que nos marcamos en el capítulo 1 de esta tesis podemos concluir que:

1. El primer objetivo que nos marcamos fue el de estudiar el estado del arte del tema para establecer un marco apropiado en el que desarrollar esta tesis. Decir que este objetivo ha sido alcanzado totalmente, sinceramente sería mentir descaradamente. La razón es sencilla, investigar en un tema, en el cual está en el momento de mayor producción científica de su historia, es físicamente imposible estar al tanto de todo lo que se publica/investiga al respecto, entre otras cosas por ser un tema que abarca infinitud de técnicas y que se investiga en la inmensa mayoría de los centros de investigación que se dedican al mundo de la lingüística computacional. Por lo tanto podemos que decir que se ha cubierto parcialmente, pero sí lo más esencial, que era dejar claro el marco de trabajo, y obtener los conocimientos suficientes para poder haber llevado a cabo todo el trabajo aquí expuesto.

2. En cuanto al objetivo de desarrollar modelos de traducción más potentes y versátiles añadiendo información contextual a los modelos léxicos, basándonos en el marco que nos proporciona la máxima entropía, podemos decir que se ha cubierto en su totalidad, incluso en mayor medida de lo que se pretendía en un principio.

3. Lo mismo podemos decir en cuanto a lo que algoritmos de búsqueda se refiere, pues en un principio fue un reto y afortunadamente ha dado el fruto esperado, aunque debemos decir que la calidad de las traducciones obtenidas está lejos de lo que se podría llamar una traducción de calidad.

Con respecto al objetivo adicional de brindar un texto de referencia en cuanto a traducción automática estadística se refiere, que a pesar de no tratarse puramente de un trabajo de investigación, no lo es por ello menos importante, ambicioso, e interesante desde nuestro punto de vista. Desafortunadamente no estamos en disposición de emitir un juicio crítico y objetivo al respecto. Por ello, debemos de conformarnos con nuestro nivel de satisfacción, que pecando de modestia nos atrevemos a decir que es alto. De cualquier forma habrá que esperar cierto tiempo para que sean los futuros lectores los que establezcan un juicio mucho más objetivo. Que así sea.

En los siguiente apartados exponemos con un poco más de detenimiento las aportaciones más relevantes de este trabajo de tesis.

Aportaciones de los modelos de traducción basados en máxima entropía

En este apartado de la tesis hemos definido, desarrollado e integrado de modelos léxicos dependientes del contexto, siguiendo la aproximación por máxima

entropía, en el aprendizaje de modelos estadísticos de alineamiento clásicos, obteniendo, en general, mejores resultados a nivel de calidad de los modelos obtenidos.

Como aportaciones a destacar tenemos:

- Hemos definido modelos léxicos refinados en función de la información contextual en que un par dado de palabras, una la traducción de la otra, aparecen en un corpus alineado a nivel de palabra. Se ha proporcionado un método de extracción y selección de características relevantes al proceso a modelar, así como la generación automática de eventos de entrenamiento, para su posterior utilización en el aprendizaje de los parámetros de dichos modelos.

- Hemos evaluado la calidad de dichos modelos léxicos en términos de entropía, demostrando que son mejores que los modelos léxicos convencionales.

- Hemos propuesto dos nuevos métodos de cómo integrar los modelos léxicos dependientes del contexto dentro del algoritmo de entrenamiento (algoritmo EM) de los modelos estadísticos de alineamiento de IBM, y demostrado su funcionamiento en términos de calidad en los alineamientos obtenidos con las nuevas aproximaciones.

- Hemos comparado la calidad de los alineamientos obtenidos con dicha integración con los obtenidos mediante los modelos convencionales, obteniendo siempre mejores resultados.

- Hemos implementado un algoritmo de repuntuación de hipótesis para utilizar dichos modelos, como modelos desambiguadores, en una fase de postproceso para mejorar sistemas de traducción automática.

Aportaciones de los algoritmos de búsqueda

Con respecto a esta parte de la tesis, podemos decir que hemos aportado distintas formas de solventar el problema de la búsqueda en la traducción automática estadística.

Más concretamente podemos destacar:

- Hemos propuesto, diseñado, implementado y estudiado una familia de algoritmos de búsqueda basados en la técnica de programación dinámica, y demostrado su aplicabilidad a tareas de dominio restringido.

- Hemos implementado la gran mayoría de algoritmos de búsqueda basados en pila propuestos en la bibliografía. También hemos realizado una comparativa entre ellos destacando las principales características de cada uno. Para todos ellos hemos expuesto todas las optimizaciones descritas

en distintas publicaciones relacionadas con el tema. Por otra parte hemos propuesto soluciones para reducir o eliminar los errores de búsqueda inherentes de dichos algoritmos.

- Hemos desarrollado, implementado, y propuesto mejoras a un algoritmo básico basado en técnicas de búsqueda local propuesto en la bibliografía, obteniendo una tercera familia de algoritmos de búsqueda, los algoritmos voraces. Para ellos, hemos propuesto optimizaciones no heurísticas, obteniendo resultados sorprendentes en cuanto a eficiencia sin mermar las prestaciones.

En todos los casos, para todos los algoritmos propuestos y estudiados, hemos realizado un estudio de su complejidad teórica. Del mismo modo hemos realizado un estudio experimental de la influencia de los parámetros de optimización en la eficiencia y eficacia de dichos algoritmos; y hemos llevado a cabo experimentos reales de traducción para tareas de diferente nivel de complejidad.

8.3 Conclusiones finales

De todo el trabajo desarrollado en esta tesis podemos concluir que:

- La máxima entropía nos proporciona un marco de trabajo que nos permite con cierta facilidad obtener modelos de traducción mejorados. Concretamente, es relativamente sencillo imponer restricciones adicionales a los modelos involucrados en un sistema de traducción automática estadística. Estas restricciones tratan de capturar ciertos fenómenos lingüísticos que podamos considerar interesantes en el proceso de modelado, y también permiten incluir información de tipo sintáctica y semántica de los lenguajes naturales. Esta propiedad del modelado por máxima entropía erige a este paradigma como uno de los más prometedores en el campo de la traducción automática estadística.

- A nuestro juicio, el asunto del entrenamiento de los modelos de traducción, se puede considerar como el principal problema de la traducción automática estadística, por encima del algoritmo que estemos considerando, ya que los errores del modelo han superado en todos los casos a los errores de búsqueda (siempre considerando la forma en que se han evaluado los resultados: con una única frase de referencia; aunque hay que recordar que muchos errores del modelo corresponden a frases incorrectas tanto sintáctica como semánticamente). Por ello, consideramos que se debería dedicar más tiempo al estudio del entrenamiento de los modelos de traducción utilizados, aunque esto pueda llegar a plantearse como un problema de optimización dada la gran cantidad de parámetros que debe de ajustarse.

- El problema de la búsqueda es un problema difícil de resolver en la práctica de acuerdo a la forma en que se han establecido las relaciones estructurales entre frases de lenguajes fuente y destino. Problema que a pesar de todo, al menos para los modelos que aquí se han estudiado, aun está por resolver de forma eficiente para obtener sistemas de traducción FAHQT (*Full Automatic High Quality Translation*), abriendo al mismo tiempo un inmenso abanico de posibilidades de continuación en este campo.

  De este modo las herramientas de traducción que se han desarrollado en esta trabajo, están muy lejos de poder ser utilizadas con un mínimo de fiabilidad como aplicación totalmente automática, aunque no obstante, otro modo de uso consiste en emplearla como soporte para hacer una traducción de calidad por parte de un experto humano.

- Otro aspecto a destacar con respecto a los algoritmos de búsqueda es que en principio solamente son susceptibles de ser utilizados con tareas sencillas y de dominio restringido. Como hemos comprobado experimentalmente, estos algoritmos aplicados a tareas grandes como HANSARDS, no podrían ser utilizados de forma práctica, aunque si con cierta garantía en sistemas de traducción en los que el tiempo de respuesta no sea un factor determinante.

- Otro aspecto digno de comentar, es la forma en que se hace la evaluación de la calidad de las traducciones que se generan. Los mecanismos completamente automáticos descritos en el capítulo 2 no están exentos de inconvenientes, que sólo se pueden resolver usando evaluadores humanos. Sin embargo, esta solución es a su vez tremendamente lenta y puede costar mucho dinero; aunque no obstante, en principio nos parece necesaria frente al excesivamente estricto criterio de comparar cada frase generada con una única frase de referencia.

- Debemos dejar claro que los resultados obtenidos con los algoritmos de traducción aquí expuestos están lejos de los alcanzados por otros sistemas estadísticos de traducción. En este sentido hay que matizar que en ningún momento hemos realizado ningún tipo de preproceso a los corpus utilizados en la experimentación.

- Por último destacar que el desarrollo e implantación de sistemas estadísticos de traducción, para nuevas tareas a tratar, teóricamente consiste simplemente en llevar a cabo un nuevo entrenamiento de los modelos de traducción y lenguaje para dicha tarea. Esto en realidad no es del todo cierto, pues la experiencia nos ha dicho que se necesita un tiempo no despreciable para ajustar los parámetros de entrenamiento y de los algoritmos de búsqueda para obtener resultados competitivos.

8.4 Trabajos futuros

En cuanto a las ideas que tenemos en mente a desarrollar en el futuro para continuar al investigación en la línea marcada en esta tesis, cabe destacar:

- En cuanto al modelado por máxima entropía nuestro plan para el futuro pasa por incluir características más discriminantes, como por ejemplo dependencias con otras palabras fuente y destino, etiquetas POS, constituyentes sintácticos, e información que pueden aportar analizadores sintácticos disponibles. También entra dentro de nuestros planes diseñar modelos por ME para los modelos de fertilidad y de distorsión, los cuales permitirán integrar fácilmente mayor número de dependencias, como modelos de distorsión/alineamiento de segundo orden sin caer en problemas de estimación de gran cantidad de parámetros como ocurre con los modelos convencionales. También esperamos obtener mejoras realizando un modelado específico para las palabras raras (o infrecuentes), las cuales no hemos tenido en cuenta en los modelos por ME aquí presentados.

- Estudiar con detenimiento el entrenamiento de los modelos de traducción basados en relaciones estructurales a nivel de palabra para intentar mejorar la calidad de los algoritmos de búsqueda propuestos. Del mismo modo, pretendemos desarrollar modelos de traducción basados en grupos de palabras, modelos que están arrojando los mejores resultados de traducción, no solo en la aproximación estadística sino en la traducción automática en general.

- Teniendo en cuenta el punto anterior, también pasa por nuestra mente adaptar los algoritmos de búsqueda para que puedan trabajar con modelos de traducción basados en grupos de palabras. Del mismo modo pretendemos incluir la aproximación por máxima entropía a dichos modelos por el gran potencial que pueden proporcionar. En este último caso, destacar que los algoritmos voraces son unos ideales candidatos a poder tratar con dichos modelos.

- Extender los algoritmos de búsqueda voraces utilizando técnicas metaheurísticas y bio-inspiradas, las cuales están siendo utilizadas con éxito en la resolución de problemas clásicos de optimización combinatoria.

- Evidentemente, también esta dentro de nuestros objetivos el extender los algoritmos de programación dinámica y de pila para los modelos HMM y 5.

- También ha pasado por nuestra mente el implementar versiones paralelas de los algoritmos de búsqueda, dado que por sus características son susceptibles de ser paralelizados con éxito. Esto unido a la disponibilidad

existente de cantidad de máquinas multiprocesador o entornos multicomputador hace atractiva esta idea. En este sentido podríamos ir incluso un poco más lejos, es decir, utilizar la moderna tecnología *grid* para realizar traducciones masivas de documentos. Esto último siempre y cuando la traducción se pueda llevar a cabo de manera completamente desacoplada y no se requiera un tiempo de respuesta razonablemente rápido. Un campo al que se podría aplicar esta técnica es al de la traducción de manuales técnicos de usuario.

- Por último comentar que también entra dentro de nuestros objetivos integrar los sistemas de traducción en interfaces de usuario reales para dotarlos de mayor versatilidad y uso, sobre todo teniendo en cuenta a gente discapacitada con problemas de accesibilidad.

A la vista de todo esto, queda patente la infinidad de posibilidades de continuación que se abren con este trabajo.

8.5 Publicaciones relacionadas con la tesis

Cabe destacar que durante el desarrollo de la investigación llevada a cabo, hasta el momento de la presentación de esta memoria se han publicado una serie de artículos en destacados foros de debate relacionados con el tema, que, en cierta medida, avalan el contenido de este trabajo. Estas publicaciones, por orden cronológico son:

- I. García-Varea, F. Casacuberta: Maximum entropy modeling: A suitable framework to learn context-dependent lexicon models for statistical machien translation. *Machine Learning*, Vol., 2004. Pendiente de aceptación.

- I. García-Varea, F. Casacuberta: A search procedure for statistical translation. In *Procs. of the VII National Symposium on Pattern Recognition and Image Analysis*, pp. 199–204, Barcelona, Spain, May 1997.

- I. García-Varea, F. Casacuberta: Statistical translation in limited domain tasks. In *Procs. of the ESSLI'98, Workshop on Machine Translation*, pp. 34–42, Saarbruecken, Germany, Aug. 1998.

- I. García-Varea, F. Casacuberta, H. Ney: An iterative, DP-based search algorithm for statistical machine translation. In *Proc. of the Int. Conf. on Spoken Language Processing (ICSLP'98)*, pp. 1235–1238, Sydney, Australia, Nov. 1998.

- I. García-Varea, A. Sanchis, F. Casacuberta: A new approach to speech-input statistical translation. In *Procs. of the International Conference*

on Pattern Recognition (ICPR'2000), Vol. 3, pp. 94–97, Barcelona, Spain, Sept. 2000. IEEE.

- I. García-Varea, F. Casacuberta. *Word Categorization in Statistical Translation*, pp. 212–220. IOS Press, 2000.

- I. García-Varea, F. Casacuberta: Search algorithms for statistical machine translation based on dynamic programming and pruning techniques. In *Proc. of Machine Translation Summit VIII*, pp. 115–120, Santiago de Compostela, Spain, Sept. 2001.

- I. García-Varea, F.J. Och, H. Ney, F. Casacuberta: Refined lexicon models for statistical machine translation using a maximum entropy approach. In *Proc. of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 204–211, Toulouse, France, July 2001.

- I. García-Varea, F.J. Och, H. Ney, F. Casacuberta. Efficient integration of maximum entropy lexicon models within the training of statistical alignment models. In C. Richardson, editor, *Machine Translation: From research to real users*, Lecture Notes in Artificial Intelligence, pp. 161–168. Springer-Verlag, 2002. The Association for Machine Translation in the Americas AMTA-2002 Conference. Tiburon, California.

- I. García-Varea, F.J. Och, H. Ney, F. Casacuberta: Improving alignment quality in statistical machine translation using context-dependent maximum entropy models. In *COLING '02: The 19th Int. Conf. on Computational Linguistics*, Vol. 2, pp. 1051–1057, Taipei, Taiwan, Aug. 2002.

- D. Ortíz, I. García-Varea, F. Casacuberta. An empirical comparison of stack-based decoding algorithms for statistical machine translation. In *New Advance in Computer Vision*, Lecture Notes in Computer Science. Springer-Verlag, 2003. 1st Iberian Conference on Pattern Recongnition and Image Analysis -IbPRIA2003- Mallorca. Spain. June.

- D. Ortíz, I. García-Varea, F. Casacuberta, A. Lagarda, J. González: On the use of statistical machine translation techniques within a memory-based translation system (ametra). In *Proc. of Machine Translation Summit IX*, pp. 115–120, New Orleans, USA, Sept. 2003.

Bibliografía

[Abeitua 01] J. Abeitua. La traducción automática: presente y futuro. http://www.serv-inf.deusto.es/abaitua/konzeptu/ta/ta97.htm, jun 2001.

[Ahrenberg et al. 00] L. Ahrenberg, M. Merkel, H.A. Sagvall, J. Tiedemann: Evaluation of word alignment systems. In *Proc. of the Second Int. Conf. on Language Resources and Evaluation (LREC)*, pp. 1255–1261, Athens, Greece, May 2000.

[Al-Onaizan et al. 99] Y. Al-Onaizan, J. Curin, M. Jahr, K. Knight, J.D. Lafferty, I.D. Melamed, D. Purdy, F.J. Och, N.A. Smith, D. Yarowsky. Statistical machine translation, final report, JHU workshop, 42 pages, Sept. 1999. `http://www.clsp.jhu.edu/ws99/projects/mt/final\_report/mt-final-report.%ps`.

[Alshawi 96a] H. Alshawi: Head automata for speech translation. In *Proc. of the Int. Conf. on Spoken Language Processing*, Vol. 4, pp. 2360–2363, Philadelphia, PA, 1996.

[Alshawi 96b] H. Alshawi. Head automata and bilingual tiling: Translation with minimal representations. Preprint-Server für Computational Linguistics, 1996.

[Alshawi and Xiang 97] H. Alshawi, F. Xiang: English-to-mandarin speech translation with head transducers. In *Spoken Language Translation Workshop (SLT-97)*, pp. 54–60, Madrid (SPAIN), July 1997.

[Amengual et al. 00] J. Amengual, J. Benedí, F. Casacuberta, M. Castaño, A. Castellanos, V. Jiménez, D. Llorens, A. Marzal, M. Pastor, F. Prat, E. Vidal, J. Vilar: The eutrans-i speech translation system. *Machine Translation*, Vol. 1, 2000.

[Arnold and des Tombe 82] D. Arnold, L. des Tombe. Basic theory and methodology in eurotra. In S. Nirenburg, editor, *Machine Translation: Theoretical and Methodological Issues*, pp. 114–135. Cambridge University Press, Cambridge, 1982.

[Arnold et al. 94] D. Arnold, L. Balkan, S. Meijer, L.L. Humphreys, L. Sadler: *Machine Translation: an Introductory Guide*. Blackwells-NCC, London, Great Britain, 1994.

[Ban and Feigenbann 81] A. Ban, A. Feigenbann, editors: *The handbook of Artificial Inteligence*. Pitman, 1981.

[Bar-Hillel 60] Y. Bar-Hillel: The present state of automatic translation of languages. *Advances in Computers*, Vol. 1, pp. 91–163, 1960.

[Baum 72] L.E. Baum: An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, Vol. 3, pp. 1–8, 1972.

[Bennett and Slocum 85] W. Bennett, J. Slocum: The rlc mahine translation system. *Computational Linguistics*, Vol. 1, pp. 91–163, 1985.

[Berger et al. 94] A.L. Berger, P.F. Brown, S.A. Della Pietra, V.J. Della Pietra, J.R. Gillett, J.D. Lafferty, H. Printz, L. Ureš: The Candide system for machine translation. In *Proc. ARPA Workshop on Human Language Technology*, pp. 157–162, Plainsboro, NJ, March 1994.

[Berger et al. 96a] A.L. Berger, P.F. Brown, S.A. Della Pietra, V.J. Della Pietra, J.R. Gillett, A.S. Kehler, R.L. Mercer. Language translation apparatus and method of using context-based translation models. United States Patent, No. 5510981, 75 pages, April 1996.

[Berger et al. 96b] A.L. Berger, S.A. Della Pietra, V.J. Della Pietra: A maximum entropy approach to natural language processing. *Computational Linguistics*, Vol. 22, No. 1, pp. 39–72, March 1996.

[Billmeyer 82] R. Billmeyer: Zu den linguistischen grundslagen von systran. *Multilingua*, Vol. 2, No. 1, pp. 83–96, 1982.

[Bishop et al. 75] Y. Bishop, S. Fienberg, P. Holland: *Discrete Multivariate Analysis*. MIT press, Cambridge, MA, 1975.

[Boitet et al. 82] C. Boitet, P. Guillaume, M. Quezel-Ambrunaz: Implementation and conversational environment of ariane 78.4: an integrated system for automated translation and human revision. In *Procs. of the 9th International Conference on Computational Linguistics*, pp. 19–27, Prague, 1982.

[Brown 95] D. Brown: A note on approximations to discrete probability distributions. *Information and Control*, Vol. 2, pp. 386–392, 1995.

[Brown et al. 90] P.F. Brown, J. Cocke, S.A. Della Pietra, V.J. Della Pietra, F. Jelinek, J.D. Lafferty, R.L. Mercer, P.S. Roossin: A statistical approach to machine translation. *Computational Linguistics*, Vol. 16, No. 2, pp. 79–85, June 1990.

[Brown et al. 91a] P.F. Brown, V.J. Della Pietra, S.A. Della Pietra, , R.L. Mercer: Word sense disambiguation using statistical methods. In *Procs. of 29th Annual Meeting of the ACL*, pp. 264–270, Berkeley, CA, jun 1991.

[Brown et al. 91b] P.F. Brown, J. Lai, R. Mercer: Aligning sentences in parallel corpora. In *Procs. of 29th Annual Meeting of the ACL*, pp. 169–176, Berkeley, CA, jun 1991.

[Brown et al. 92] P.F. Brown, V.J. Della Pietra, P.V. deSouza, J.C. Lai, R.L. Mercer: Class-based n-gram models of natural language. *Computational Linguistics*, Vol. 18, No. 4, pp. 467–479, 1992.

[Brown et al. 93] P.F. Brown, S.A. Della Pietra, V.J. Della Pietra, R.L. Mercer: The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, Vol. 19, No. 2, pp. 263–311, 1993.

[Brown et al. 94] P. Brown, S. Chen, V.D. Pietra, S.D. Pietra, A. Keller, R. Mercer: Automatic speech recognition in machine translation. *Computer Speech and Language*, Vol. 8, No. 1, pp. 177–187, April 1994.

[Canals-Marote et al. 01] R. Canals-Marote, A. Esteve-Guillén, A. Garrido-Alenda, M. Guardiola-Savall, A. Iturraspe-Bellver, S. Montserrat-Buendia, S. Ortiz-Rojas, H. Pastor-Pina, P. Pérez-Antón, M. Forcada: The spanish-catalan machine translation system internostrum. In *Proc. of Machine Translation Summit VIII*, pp. 73–76, Santiago de Compostela, Spain, Sept. 2001.

[Casacuberta 00] F. Casacuberta. Inference of finite-state transducers by using regular grammars and morphisms. In A. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, Vol. 1891 of *Lecture Notes in Computer Science*, pp. 1–14. Springer-Verlag, 2000. 5th International Colloquium Grammatical Inference -ICGI2000-. Lisboa. Portugal. Septiembre.

[Casacuberta and Vidal 88] F. Casacuberta, E. Vidal, editors: *Reconocimiento Automática del Habla.* Marcombo, 1988.

[Castaño et al. 97] A. Castaño, F. Casacuberta, V. Vidal: Machine translation using neural networks and finite-state models. In *Procs. of thr 7th International Conference on Theoretical and Methodological Issues in Machine Translation*, pp. 160–167, Santa Fe, New Mexico, 1997.

[Castellanos et al. 94] A. Castellanos, I. Galiano, E. Vidal. Application of OSTIA to machine translation tasks. In R.C. Carrasco, J. Oncina, editors, *Grammatical Inference and Applications, Proc. of 2nd ICGI*, Vol. 862 of *Lecture Notes in Computer Science*, pp. 93–105. Springer-Verlag, Alicante, Spain, 1994.

[Chang and Chen 94] J.S. Chang, H.C. Chen: Using partially aligned parallel text and part-of-speech information in word alignment. In *First Conf. of the Association for Machine Translation in the Americas (AMTA 94)*, pp. 16–23, Columbia, MD, 1994.

[Charniak 99] E. Charniak. A maximum-entropy-inspired parser. Technical Report CS-99-12, 1999.

[Chen and Goodman 96] S.F. Chen, J. Goodman: An empirical study of smoothing techniques for language modeling. In A. Joshi, M. Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pp. 310–318, San Francisco, 1996. Morgan Kaufmann Publishers.

[Cover and Thomas 91] T. Cover, J. Thomas: *Elements of Information Theory.* John Wiley and Sons., New York, 1991.

[Dagan et al. 93] I. Dagan, K.W. Church, W.A. Gale: Robust bilingual word alignment for machine aided translation. In *Proc. of the Workshop on Very Large Corpora*, pp. 1–8, Columbus, OH, June 1993.

[Darroch and Ratcliff 72] J.N. Darroch, D. Ratcliff: Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics*, Vol. 43, pp. 1470–1480, 1972.

[Della Pietra et al. 94] S.A. Della Pietra, V.J. Della Pietra, J.R. Gillett, J.D. Lafferty, H. Printz, L. Ureš: Inference and estimation of a long-range trigram model. In *Second Int. Symposium on Grammatical Inference*, Alicante, Spain, 1994.

[Della Pietra et al. 97] S.A. Della Pietra, V.J. Della Pietra, J.D. Lafferty: Inducing features in random fields. *IEEE Trans. on Pattern Analysis and Machine Inteligence*, Vol. 19, No. 4, pp. 380–393, July 1997.

[Dempster et al. 77] A.P. Dempster, N.M. Laird, D.B. Rubin: Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statist. Soc. Ser. B*, Vol. 39, No. 1, pp. 1–22, 1977.

[Dorr 93] B. Dorr: *Machine Translation.* MIT Press, Cambridge, MA, 1993.

[Duda et al. 00] R.O. Duda, P.E. Hart, D.G. Stork: *Pattern Classification.* John Wiley and Sons, New York, NY, 2nd edition, 2000.

[Fabra 99] U.P. Fabra. Programa salt de traducción automática español-catalán. http://www.upf.es/grec/gl/cast/salt.htm, jun 1999.

[Foster 00a] G. Foster: Incorporating position information into a maximum entropy/minimum divergence translation model. In *Fourth Conf. on Computational Language Learning (CoNLL)*, pp. 37–52, Lisbon, Portugal, Sept. 2000.

[Foster 00b] G. Foster: A maximum entropy/minimum divergence translation model. In *Proc. of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 37–44, Hong Kong, Oct. 2000.

[Fung and Church 94] P. Fung, K.W. Church: K-vec: A new approach for aligning parallel texts. In *COLING '94: The 15th Int. Conf. on Computational Linguistics*, Aug. 1994.

[Fung and McKeown 94] P. Fung, K. McKeown: Aligning noisy parallel corpora across language groups: Word pair feature matching by dynamic warping. In *First Conf. of the Association for Machine Translation in the Americas (AMTA 94)*, pp. 81–88, Columbia, MD, Oct. 1994.

[Gale et al. 92] B. Gale, K. Church, D. Yarowsky: Work on statistical methods for word sense disambiguation. In *Procs. of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, pp. 54–60, Cambridge, MA, 1992.

[García-Varea and Casacuberta 98] I. García-Varea, F. Casacuberta: Statistical translation in limited domain tasks. In *Procs. of the ESSLI'98, Workshop on Machine Translation*, pp. 34–42, Saarbrücken, Germany, Aug. 1998.

[García-Varea and Casacuberta 01] I. García-Varea, F. Casacuberta: Search algorithms for statistical machine translation based on dynamic programming and pruning techniques. In *Proc. of Machine Translation Summit VIII*, pp. 115–120, Santiago de Compostela, Spain, Sept. 2001.

[García-Varea et al. 98] I. García-Varea, F. Casacuberta, H. Ney: An iterative, DP-based search algorithm for statistical machine translation. In *Proc. of the Int. Conf. on Spoken Language Processing (ICSLP'98)*, pp. 1235–1238, Sydney, Australia, Nov. 1998.

[García-Varea et al. 99] I. García-Varea, A. Sanchis, F. Casacuberta. On speech input statistical translation. Unpublished paper, Dec. 1999.

[García-Varea et al. 00] I. García-Varea, A. Sanchis, F. Casacuberta: A new approach to speech-input statistical translation. In *Procs. of the International Conference on Pattern Recognition (ICPR'2000)*, Vol. 3, pp. 94–97, Barcelona, Spain, Sept. 2000. IEEE.

[García-Varea et al. 01] I. García-Varea, F.J. Och, H. Ney, F. Casacuberta: Refined lexicon models for statistical machine translation using a maximum entropy approach. In *Proc. of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 204–211, Toulouse, France, July 2001.

[García-Varea et al. 02a] I. García-Varea, F.J. Och, H. Ney, F. Casacuberta. Efficient integration of maximum entropy lexicon models within the training of statistical alignment models. In C. Richardson, editor, *Machine Translation: From research to real users*, Lecture Notes in Artificial Intelligence, pp. 161–168. Springer-Verlag, 2002. The Association for Machine Translation in the Americas AMTA-2002 Conference. Tiburon, California.

[García-Varea et al. 02b] I. García-Varea, F.J. Och, H. Ney, F. Casacuberta: Improving alignment quality in statistical machine translation using context-dependent maximum entropy models. In *COLING '02: The 19th Int. Conf. on Computational Linguistics*, Vol. 2, pp. 1051–1057, Taipei, Taiwan, Aug. 2002.

[Germann et al. 01] U. Germann, M. Jahr, K. Knight, D. Marcu, K. Yamada: Fast decoding and optimal decoding for machine translation. In *Proc. of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 228–235, Toulouse, France, July 2001.

[Guiasu and Shenitzer 94] S. Guiasu, A. Shenitzer: The principle of maximum entropy. *The Mathematical Inteligencer*, Vol. 7, No. 1, 1994.

[Hobbs 92] J. Hobbs. Machine translation. Technical report, DARPA. Software and Intelligent Systems Technology Office, Feb. 1992.

[Huang and Choi 00] J.X. Huang, K.S. Choi: Chinese–Korean word alignment based on linguistic comparison. In *Proc. of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 392–399, Hong Kong, Oct. 2000.

[Huang et al. 01] X. Huang, A. Acero, H. Hon: *Spoken Language Processing: A guide to Theory, Algorithm, and System Development.* Prentice Hall, Upper Saddle River, NJ 07458, 2001.

[Hutchins and Somers 92] W.J. Hutchins, H.L. Somers: *An Introduction to Machine Translation.* Academic Press, Cambridge, MA, 1992.

[Ide and Véronis 98] N. Ide, J. Véronis: Introduction to the special issue on word sense disambiguation: The state of the art. *Computational Linguistics*, Vol. 1, No. 24, 1998.

[Isabelle 92] P. Isabelle, editor. *Fourth International Conference on Theoretical and Methodological Issues in Machine Translation. Empiricist vs. Rationalist Methods in MT*, Montréal: CCRIT-CWARC, 1992.

[Jardino and Adda 93] M. Jardino, G. Adda: Automatic word classification using simulated annealing. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, Vol. 2, pp. 41–44, Minneapolis, MN, 1993.

[Jaynes 90] E. Jaynes. Notes on present status and future prospects. In W. Grandy, L. Schick, editors, *Maximum Entropy and Bayesian Methods*, pp. 1–13. Kluwer, 1990.

[Jelinek 69] F. Jelinek: A fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, Vol. 13, pp. 675–685, 1969.

[Jelinek et al. 90] F. Jelinek, R. Mercer, S. Roukos: Classifying words for improved statistical language models. In *Procs. of the ICASSP'90*, pp. 621–624, Alburquerque, NM, USA, 1990.

[Jones and Rusk 00] D.A. Jones, G.M. Rusk: Toward a scoring function for quality-driven machine translation. In *COLING '00: The 18th Int. Conf. on Computational Linguistics*, pp. 376–382, Saarbrücken, Germany, Aug. 2000.

[Jones and Somers 95] D.B. Jones, H.L. Somers: Automatically determining bilingual vocabulary from noisy bilingual corpora using variable bag estimation. In *Recent Advances in Natural Language Processing*, pp. 81–86, Sept. 1995.

[Jurafsky and Martin 00] D. Jurafsky, J. Martin: *Speech and Language Processing: An introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall, Upper Saddle River, NJ 07458, 2000.

[Kay and Röscheisen 93] M. Kay, M. Röscheisen: Text-translation alignment. *Computational Linguistics*, Vol. 19, No. 1, pp. 121–142, 1993.

[Kay et al. 94] M. Kay, J. Gawron, P. Norving. Verbmobil: A translation system fo face-to-face dialog. In *CLSI*, number 33 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

[Ker and Chang 97] S.J. Ker, J.S. Chang: A class-based approach to word alignment. *Computational Linguistics*, Vol. 23, No. 2, pp. 313–343, 1997.

[Khudanpur and Wu 99] S. Khudanpur, J. Wu: A maximum entropy language model to integrate n-grams and topic dependencies for conversational speech recognition. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 553–556, Phoenix, USA, March 1999.

[Khudanpur and Wu 00] S. Khudanpur, J. Wu: Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling. *Computer, Speech and Language*, Vol. 14, pp. 355–372, Oct. 2000.

[Kneser and Ney 93a] R. Kneser, H. Ney. Forming word classes by statistical clustering for statistical language modelling. In R. Kohler, B.B. Rieger, editors, *Contributions to Quantitative Linguistics*, pp. 221–226. Kluwer Academic Publishers, 1993.

[Kneser and Ney 93b] R. Kneser, H. Ney: Improved clustering techniques for class-based statistical language modelling. In *European Conf. on Speech Communication and Technology*, pp. 973–976, Berlin, Germany, Sept. 1993.

[Knight 99a] K. Knight: Decoding complexity in word-replacement translation models. *Computational Linguistics*, Vol. 25, No. 4, pp. 607–615, 1999.

[Knight 99b] K. Knight. A statistical machine translation tutorial workbook, 35 pages, Aug. 1999. `http://www.isi.edu/natural-language/mt/wkbk.rtf`.

[Knight and Al-Onaizan 98] K. Knight, Y. Al-Onaizan: Translation with finite-state devices. In *4th AMTA*, 1998.

[Koncar and Guthrie 97] N. Koncar, G. Guthrie. *A Natural-Language-Translation Neural Nerwork*, pp. 219–228. London: UCL Press, 1997.

[Levenstein 65] V. Levenstein: Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, Vol. 10, No. 8, pp. 707–710, 1965.

[Levow 97] G.A. Levow. Corpus-based techniques for word sense disambiguation. Technical Report AIM-1637, 20, 1, 1997.

[Llorens et al. 95] D. Llorens, V. Jiménez, J. Sánchez, E. Vidal, H.Rulot: Atros, an automatically trainable continuous-speech recognition system for limited-domain tasks. In *Procs. of the VI National Symposium on Pattern Recognition and Image Analysis*, pp. 478–483, Barcelona, Spain, May 1995.

[Maas 77] H. Maas. Ergegnisse der satzanalyse und transfoomationelle synthese im saarbrücker übersetzungssystem susy, sfb 100: Elektronische sprachforschung. Technical report, Universität des Saarlandes, Saarbrücken, 1977.

[Macklovitch and Hannan 96] E. Macklovitch, M.L. Hannan: Line 'em up: Advances in alignment technology and their impact on translation support tools. In *2nd Conf. of the Association for Machine Translation in the Americas (AMTA 96)*, Montreal, Canada, 1996.

[Manning and Schütze 01] C.D. Manning, H. Schütze: *Foundations of Statistical Natural Language Processing.* MIT Press, Cambridge, Massachesetts 02142, 2001.

[Mansilla 02] J.A.G. Mansilla. Algoritmos de búsqueda basados en programación dinámica para la traducción automática estadística. Proyecto fin de carrera, Dpto. de Informática, Univ. de Castilla-La Mancha, Albacete, Spain, Sept. 2002.

[Martin et al. 99] S. Martin, H. Ney, J. Zaplo: Smoothing methods in maximum entropy language modeling. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 545–548, Phoenix, AR, March 1999.

[Martin et al. 00] S. Martin, H. Ney, C. Hamacher: Maximum entropy language modeling and the smoothing problem. *IEEE Trans. on Speech and Language Processing*, Vol. 8, No. 5, pp. 626–632, Sept. 2000.

[McLean 92] I. McLean: Example-based machine translation using connectionist matching. In [Isabelle 92], pp. 35–43.

[Melamed 97] I.D. Melamed: A word-to-word model of translational equivalence. In *Proc. 35th Annual Conf. of the Association for Computational Linguistics*, Madrid, Spain, June 1997.

[Mooney 96] R.J. Mooney. Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning. In E. Brill, K. Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 82–91. Association for Computational Linguistics, Somerset, New Jersey, 1996.

[Murray Jr. 66] H. Murray Jr.: *Methods for Satisfying the Needs of the Scientist and the Engineer for Scientific and Technical Communication.* In a Press realease, Washington D.C., 1966.

[Nagao 84] M. Nagao. A framework of a mechanical translation between japanese and english by analogy principle. In A. Elithorn, R. Banerji, editors, *Artificial and Human Intelligence*, chapter 11, pp. 173–180. Elsevier Science Publishers, 1984.

[Ney 99] H. Ney: Speech translation: Coupling of recognition and translation. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 517–520, Phoenix, AR, March 1999.

[Ney et al. 87] H. Ney, D. Mergel, A. Noll, A. Paeseler: A data-driven organization of the dynamic programming beam search for continuous speech recognition. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 833–836, Dallas, TX, April 1987.

[Ney et al. 95] H. Ney, M. Generet, F. Wessel: Extensions of absolute discounting for language modeling. In *Proc. of the Fourth European Conf. on Speech Communication and Technology*, pp. 1245–1248, Madrid, Spain, Sept. 1995.

[Ney et al. 97] H. Ney, S. Martin, F. Wessel. Statistical language models using leaving-one-out. In S. Young, G. Bloothooft, editors, *Corpus Based Methods in Language and Speech Processing*, pp. 174–207. Kluwer, 1997.

[Ney et al. 00] H. Ney, S. Nießen, F.J. Och, H. Sawaf, C. Tillmann, S. Vogel: Algorithms for statistical translation of spoken language. *IEEE Trans. on Speech and Audio Processing*, Vol. 8, No. 1, pp. 24–36, Jan. 2000.

[Nießen et al. 98] S. Nießen, S. Vogel, H. Ney, C. Tillmann: A DP-based search algorithm for statistical machine translation. In *COLING-ACL '98: 36th Annual Meeting of the Association for Computational Linguistics and 17th Int. Conf. on Computational Linguistics*, pp. 960–967, Montreal, Canada, Aug. 1998.

[Nießen et al. 00] S. Nießen, F.J. Och, G. Leusch, H. Ney: An evaluation tool for machine translation: Fast evaluation for machine translation research. In *Proc. of the Second Int. Conf. on Language Resources and Evaluation (LREC)*, pp. 39–45, Athens, Greece, May 2000.

[Och 99] F.J. Och: An efficient method for determining bilingual word classes. In *EACL '99: Ninth Conf. of the Europ. Chapter of the Association for Computational Linguistics*, pp. 71–76, Bergen, Norway, June 1999.

[Och 00] F.J. Och. Giza++: Training of statistical translation models, 2000. `http://www-i6.informatik.rwth-aachen.de/\~och/software/GIZA++.html`.

[Och 02] F.J. Och. *Statistical Machine Translation: From Single-Word Models to Alignment Templates*. Ph.D. thesis, Computer Science Department, RWTH Aachen, Germany, October 2002.

[Och and Ney 00a] F.J. Och, H. Ney: A comparison of alignment models for statistical machine translation. In *COLING '00: The 18th Int. Conf. on Computational Linguistics*, pp. 1086–1090, Saarbrücken, Germany, Aug. 2000.

[Och and Ney 00b] F.J. Och, H. Ney: Improved statistical alignment models. In *Proc. of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 440–447, Hong Kong, Oct. 2000.

[Och and Ney 02] F.J. Och, H. Ney: Discriminative training and maximum entropy models for statistical machine translation. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, PA, July 2002.

[Och and Ney 03] F.J. Och, H. Ney: A systematic comparison of various statistical alignment models. *Computational Linguistics*, Vol. 29, No. 1, pp. 19–51, March 2003.

[Och et al. 99] F.J. Och, C. Tillmann, H. Ney: Improved alignment models for statistical machine translation. In *Proc. of the Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora*, pp. 20–28, University of Maryland, College Park, MD, June 1999.

[Och et al. 01] F.J. Och, N. Ueffing, H. Ney: An efficient A* search algorithm for statistical machine translation. In *Data-Driven Machine Translation Workshop*, pp. 55–62, Toulouse, France, July 2001.

[Papageorgiou et al. 94] H. Papageorgiou, L. Cranias, S. Piperidis: Automatic alignment in parallel corpora. In *Proc. of the 32nd Annual Conf. of the Association for Computational Linguistics*, pp. 331–333, Las Cruces, NM, 1994.

[Papineni et al. 98] K.A. Papineni, S. Roukos, R.T. Ward: Maximum likelihood and discriminative training of direct translation models. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 189–192, Seattle, WA, May 1998.

[Papineni et al. 01] K.A. Papineni, S. Roukos, T. Ward, W.J. Zhu. Bleu: a method for automatic evaluation of machine translation. Technical Report RC22176 (W0109-022), IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY, 10 pages, Sept. 2001.

[Peters and Klakow 99] J. Peters, D. Klakow: Compact maximum entropy language models. In *Proc. of the IEEE Workshop on Automatic Speech Recognition and Understanding*, Keystone, CO, Dec. 1999.

[Pierce and others 93] J. Pierce et al. Languages and machines - computers in translation and linguistics. Technical Report Technical Report, Automatic Language Processing Advisory Committee (ALPAC), Washington D.C., 1993.

[Prat 98] F. Prat. *Traducción automática en dominios restringidos: Algunos modelos susceptibles de ser aprendidos a partir de ejemplos.* Ph.D. thesis, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, 1998.

[Prat 01] F. Prat: Machine translation with grammar association: Some improvements and the loco_c model. In *Procs. of the Data-driven Machine Translation ACL'01 workshop*, pp. 71–78, Toulouse, France, July 2001.

[Quatieri 02] T.F. Quatieri: *Discrete-time Speech Signal Processing: Principles and Practice*. Prentice Hall, Upper Saddle River, NJ 07458, 2002.

[Ratnaparkhi 96] A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In E. Brill, K. Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 133–142. Association for Computational Linguistics, Somerset, New Jersey, 1996.

[Ratnaparkhi 97] A. Ratnaparkhi. A simple introduction to maximum entropy models for natural language processing, 1997.

[Ratnaparkhi 98] A. Ratnaparkhi. Maximum entropy models for natural language ambiguity resolution, 1998.

[Ratnaparkhi 99] A. Ratnaparkhi: Learning to parse natural language with maximum entropy models. *Machine Learning*, Vol. 34, pp. 151, 1999.

[Rosenfeld 96] R. Rosenfeld: A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, Vol. 10, pp. 187–228, 1996.

[Rosenfeld and Clarkson 97] R. Rosenfeld, P. Clarkson. Statistical language modelling using the cmu-cambridge toolkit. Technical report, CMU, 1997.

[Rosetta 94] M. Rosetta: *Compositional Translation*. Kluwer, Dordrecht, The Netherlands, 1 edition, 1994.

[Sato and Nagao 90] S. Sato, M. Nagao: Toward memory-based translation. In H. Karlgren, editor, *COLING '90: The 13th Int. Conf. on Computational Linguistics*, Vol. 3, pp. 247–252, 1990.

[Shin et al. 96] J.H. Shin, Y.S. Han, K.S. Choi: Bilingual knowledge acquisition from korean-english parallel corpus using alignment method (korean-english alignment at word and phrase level). In *COLING '96: The 16th Int. Conf. on Computational Linguistics*, pp. 230–235, 1996.

[Simons et al. 97] .M. Simons, H. Ney, S.C. Martin: Distant bigram language modelling using maximum entropy. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 787–790, Munich, Germany, April 1997.

[Somers 98] H. Somers: New paradigms in mt: the state of play now that the dust has settled. In *Procs. of the ESSLI'98, Workshop on Machine Translation*, pp. 22–33, Saarbrücken, Germany, Aug. 1998.

[Stolcke 97] A. Stolcke. Srilm - an extensible language modelling toolkit. Technical report, Speech Technology and Research Laboratory, SRI International, Menlo Park, CA, USA, 1997.

[Tihouin 82] B. Tihouin. The météo system. In V. Lawson, editor, *Experience of Machine Translation*, pp. 39–44. North-Holland, Amsterdam, 1982.

[Tillmann 01] C. Tillmann. *Word Re-Ordering and Dynamic Programming based Search Algorithms for Statistical Machine Translation*. Ph.D. thesis, Computer Science Department, RWTH Aachen, Germany, May 2001.

[Tillmann and Ney 96] C. Tillmann, H. Ney: Selection criteria for word trigger pairs in language modelling. In *Grammatical Inference: Learning Syntax from Sentences, 3rd International Colloquium, ICGI-96, Montpellier, France, September 25-27, 1996, Proceedings*, Vol. 1147, pp. 95–106. Springer, Berlin, 1996.

[Tillmann and Ney 97] C. Tillmann, H. Ney: Word trigger and the em algorithm. In *Proc. Workshop Computational Natural Language Learning*, pp. 117–124, Madrid, Spain, July 1997.

[Tillmann et al. 97a] C. Tillmann, S. Vogel, H. Ney, A. Zubiaga: A DP-based search using monotone alignments in statistical translation. In *Proc. 35th Annual Conf. of the Association for Computational Linguistics*, pp. 289–296, Madrid, Spain, July 1997.

[Tillmann et al. 97b] C. Tillmann, S. Vogel, H. Ney, A. Zubiaga, H. Sawaf: Accelerated DP based search for statistical translation. In *European Conf. on Speech Communication and Technology*, pp. 2667–2670, Rhodes, Greece, Sept. 1997.

[Tomás and Casacuberta 01] J. Tomás, F. Casacuberta: Monotone statistical translation using word groups. In *Procs. of the Machine Translation Summit VIII*, pp. 357–361, Santiago de Compostela, Spain, 2001.

[Tomás and Casacuberta 02] J. Tomás, F. Casacuberta: Binary feature classification for word disambigaution in statistical machine translation. In *Procs. of the 2nd International Workshop on Pattern Recognition and Information Systems*, Alicante, Spain, 2002.

[Trujillo 99] A. Trujillo: *Translation Engines: Techniques for Machine Translation*. Springer-Verlag, London, 1 edition, 1999.

[Tz-Liang and Su 02] Tz-Liang, K.K.Y. Su: A robust cross-style bilingual sentences alignment model. In *COLING '02: The 19th Int. Conf. on Computational Linguistics*, Vol. 2, pp. 2071–2077, Taipei, Taiwan, Aug. 2002.

[Vauquois 75] B. Vauquois: *La Traduction Automatique á Grenoble*. Dunod, Paris, 1 edition, 1975.

[Vidal 97] E. Vidal: Finite-state speech-to-speech translation. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, Vol. 1, pp. 111–114, Munich, Germany, April 1997.

[Vidal and Prieto 92] E. Vidal, N. Prieto: Learning language models through the ecgi method. *Speech Communication*, Vol. 11, pp. 299–309, 1992.

[Vilar 98] J. Vilar. *Aprendizaje de traductores subsecuenciales para su empleo en tareas de dominio restringido*. Ph.D. thesis, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, 1998.

[Vogel et al. 96] S. Vogel, H. Ney, C. Tillmann: HMM-based word alignment in statistical translation. In *COLING '96: The 16th Int. Conf. on Computational Linguistics*, pp. 836–841, Copenhagen, Denmark, Aug. 1996.

[Vogel et al. 00] S. Vogel, S. Nießen, H. Ney: Automatic extrapolation of human assessment of translation quality. In *2nd Int. Conf. on Language Resources and Evaluation (LREC 2000): Proc. of the Workshop on Evaluation of Machine Translation*, pp. 35–39, Athens, Greece, May 2000.

[Wahlster 00] W. Wahlster, editor: *Verbmobil: Foundations of speech-to-speech translations*. Springer Verlag, Berlin, Germany, 2000.

[Waibel et al. 91] A. Waibel, N. Jain, A. McNair, H. Saito, A. Hauptmann, J. Tebelskis: Janus: A speech-to-speech translation system using connectionist and symbolic processing strategies. In *Procs. of the IEEE Int. Conf. on Acoustic, Speech and Signal processing*, pp. 366–372, Toronto, 1991.

[Wang and Waibel 97] Y.Y. Wang, A. Waibel: Decoding algorithm in statistical translation. In *Proc. 35th Annual Conf. of the Association for Computational Linguistics*, pp. 366–372, Madrid, Spain, July 1997.

[Wang and Waibel 98] Y.Y. Wang, A. Waibel: Modeling with structures in statistical machine translation. In *COLING-ACL '98: 36th Annual Meeting of the Association for Computational Linguistics and 17th Int. Conf. on Computational Linguistics*, Vol. 2, pp. 1357–1363, Montreal, Canada, June 1998.

[Wang et al. 02] W. Wang, J.X. Huang, M. Zhou, C. Huang: Structure alignment using bilingual chunking. In *COLING '02: The 19th Int. Conf. on Computational Linguistics*, Vol. 2, pp. 1071–1077, Taipei, Taiwan, Aug. 2002.

[Weaver 55] W. Weaver. Translation. In W.N. Locke, A.D. Booth, editors, *Machine Translation of Languages: fourteen essays*, pp. 15–23. MIT Press, Cambridge, MA, 1955.

[Witkam 88] T. Witkam: Dlt: an industrial r& d project for multilingual machine translation. In *Procs.of the 12th International Conference on Computational Linguistics*, Vol. 2, pp. 756–759, Budapest, 1988.

[Wu 94] D. Wu: Aligning a parallel english-chinese corpus statistically with lexical criteria. In *Proc. of the 32nd Annual Conf. of the Association for Computational Linguistics*, pp. 80–87, Las Cruces, NM, June 1994.

[Wu 96] D. Wu: A polynomial-time algorithm for statistical machine translation. In *Proc. of the 34th Annual Conf. of the Association for Computational Linguistics (ACL '96)*, pp. 152–158, Santa Cruz, CA, June 1996.

[Wu 98] J. Wu. A maximum entropy language model with topic sensitive features, 1998.

[Wu and Khudanpur 99] J. Wu, S. Khudanpur: Combining nonlocal, syntactic and n-gram dependencies in language modeling. In *European Conf. on Speech Communication and Technology*, Vol. 5, pp. 2179–2182, Budapest, Hungary, Sept. 1999.

[Wu and Khudanpur 00] J. Wu, S. Khudanpur: Efficient training methods for maximum entropy language modeling. In *Proc. of the Int. Conf. on Spoken Language Processing*, Vol. 3, pp. 114–117, Beijing, China, Oct. 2000.

[Wu and Khudanpur 02] J. Wu, S. Khudanpur: Building a topic-dependent maximum entropy language model for very large corpora. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, Orlando, USA, May 2002.

[Zhou and Lua 98] G. Zhou, K. Lua: Word association and MI-TRigger-based language modeling. In *COLING-ACL '98: 36th Annual Meeting of the Association for Computational Linguistics and 17th Int. Conf. on Computational Linguistics*, pp. 1465–1471, 1998.

V

Apéndices

Apéndice A

Introducción a la traducción estadística con entrada de voz

En el desarrollo anterior hemos asumido que la traducción se realiza de texto a texto, o sea la frase de entrada se supone perfecta y sin errores. En el caso de traducción cuando en la entrada se considera voz en vez de texto esta asunción ya no es válida dado que nos enfrentamos al problema añadido de los errores producidos por el sistema de reconocimiento que proporcione la entrada de voz. Por lo tanto el problema de la entrada de voz es el de como integrar las probabilidades del proceso de reconocimiento dentro del proceso de traducción antes comentado.

Este problema aquí planteado se puede considerar como el proceso:

$$x_1^T \rightarrow f_1^J \rightarrow e_1^I$$

en el cual podemos diferenciar tres elementos: los vectores acústicos $x_1^T = x_1 \cdots x_t \cdots x_T$ a través del tiempo $t = 1 \cdots T$, las palabras de la frase de entrada f_1^J que serán la decodificación de los anteriores, y e_1^I que será su correspondiente traducción.

Al igual que para el caso de traducción texto a texto podemos aplicar la regla de Bayes para descomponer nuestro problema en la estimación de distintos modelos y la correspondiente búsqueda de la solución óptima. De este modo:

$$\begin{aligned}\hat{e}_1^I &= \arg\max_{e_1^I} Pr(e_1^I \mid x_1^T) = \\ &= \arg\max_{e_1^I} \{Pr(e_1^I) \cdot Pr(x_1^T \mid e_1^I)\} \\ &= \arg\max_{e_1^I} \left\{ Pr(e_1^I) \cdot \sum_{J, f_1^J} Pr(f_1^J, x_1^T \mid e_1^I) \right\} \\ &= \arg\max_{e_1^I} \left\{ Pr(e_1^I) \cdot \sum_{J, f_1^J} Pr(f_1^J \mid e_1^I) \cdot Pr(x_1^T \mid f_1^J, e_1^I) \right\}\end{aligned}$$

Finalmente esta expresión se puede aproximar por:

$$\begin{aligned}&\arg\max_{e_1^I} Pr(e_1^I \mid x_1^T) \approx \\ &\qquad \arg\max_{e_1^I} \left\{ Pr(e_1^I) \cdot \max_{f_1^J} \{Pr(f_1^J \mid e_1^I) \cdot Pr(x_1^T \mid f_1^J)\} \right\}\end{aligned}$$

asumiendo que $Pr(x_1^T \mid f_1^J, e_1^I)$ no depende de e_1^I, o sea que la frase e_1^I no ayuda a predecir los vectores acústicos (en el lenguaje de entrada) si la frase origen f_1^J es dada.

La última ecuación es una aproximación a la original dado que resolvemos el problema desde el punto de vista de una maximización en vez de una suma.

Apéndice B

Más sobre algoritmos basados en PD

B.1 Criterio de búsqueda para el modelo IBM3

Definimos la siguiente cantidad auxiliar:

$$Q\left(\mathcal{C}, \phi, \boldsymbol{\pi}, e', e, i\right)$$

donde:

- $\mathcal{C}$ es el conjunto de posiciones cubiertas de la frase de entrada cuando se ha generado la palabra e_i.
- ϕ es la fertilidad de la palabra e_i.
- $\boldsymbol{\pi}, \boldsymbol{\tau}$ son la permutación y la tupla asociadas a la palabra e_i.
- e', e palabras que ocupan las posiciones $i-1$ e i respectivamente, en la hipótesis.

La recursión se define del siguiente modo:

- Si $\phi = 0$:

$$Q\left(\mathcal{C}, \phi, \boldsymbol{\pi}, e', e, i\right) = n(\phi \mid e) \max_{e'', \phi'} \left\{p(e \mid e', e'') \cdot Q\left(\mathcal{C}, \phi', \boldsymbol{\pi}, e'', e', i-1\right)\right\}$$

- Si $\phi > 0$:

$$\begin{aligned} Q(\mathcal{C}, \phi, \boldsymbol{\pi}, e', e, i) &= n(\phi \mid e) \cdot \prod_{k=1}^{\phi} t(f_{\pi_{ik}} \mid e) \cdot d(\pi_{ik} \mid i, J) \\ &\quad \max_{e'', \phi', \boldsymbol{\pi}'} \{p(e \mid e', e'') \cdot Q(\mathcal{C} - \boldsymbol{\pi}, \phi', \boldsymbol{\pi}', e'', e', i-1)\} \end{aligned}$$

B.2 Criterio de búsqueda para el modelo IBM4

Definimos la siguiente cantidad auxiliar:

$$Q(\mathcal{C}, \phi, \boldsymbol{\pi}, e', e, i)$$

donde:

- $\mathcal{C}$ es el conjunto de posiciones cubiertas de la frase de entrada cuando se ha generado la palabra e_i.
- ϕ es la fertilidad de la palabra e_i.
- $\boldsymbol{\pi}, \boldsymbol{\tau}$ son la permutación y la tupla asociadas a la palabra e_i.
- e', e palabras que ocupan las posiciones $i-1$ e i respectivamente, en la hipótesis.

La recursión se define del siguiente modo:

- Si $\phi = 0$:

$$Q(\mathcal{C}, \phi, \boldsymbol{\pi}, e', e, i) = n(\phi \mid e) \max_{e'', \phi'} \{p(e \mid e', e'') \cdot Q(\mathcal{C}, \phi', \boldsymbol{\pi}, e'', e', i-1)\}$$

- Si $\phi > 0$:

$$\begin{aligned} Q(\mathcal{C}, \phi, \boldsymbol{\pi}, e', e, i) = n(\phi \mid e) \cdot \max_{e'', \phi', \boldsymbol{\pi}'} \quad \{ \quad & p(e \mid e', e'') \cdot \\ & \prod_{k=1}^{\phi} t(f_{\pi_{ik}} \mid e) \cdot \\ & d_{=1}(\pi_{i1} - c_{\rho_i} \mid \mathcal{A}(e_{\rho_i}), \mathcal{B}(\tau_{i1})) \cdot \\ & \prod_{k=2}^{\phi} d_{>1}(\pi_{ik} - \pi_{ik-1} \mid \mathcal{B}(\tau_{ik})) \cdot \\ & Q(\mathcal{C} - \boldsymbol{\pi}, \phi', \boldsymbol{\pi}', e'', e', i-1)\} \end{aligned}$$

Apéndice C

Estudio de los parámetros de optimización para la tarea Hansards

En este apéndice se expone un estudio de la influencia de los parámetros de optimización similar al que se ha llevado a cabo en los capítulos 5, 6 y 7 para la tarea EuTrans-I. En las tres secciones siguientes realizaremos el estudio para los tres tipos de algoritmos expuestos en la tesis.

Para ello hemos escogido 100 frases del conjunto test mostrado en la tabla 2.4 (ver sección 2.7.3, página 47) de longitudes menor o igual a 15, dada la complejidad que conlleva esta tarea. A titulo informativo, en la tabla C.1 podemos ver las características de este corpus de test.

		Francés	Inglés
Test	Frases	100	
	Palabras	830	794

Tabla C.1: Estadísticas del corpus de test utilizado en el establecimiento de los parámetros de los algoritmos de búsqueda, para la tarea Hansards.

Los modelos de traducción, que utilizaremos en toda la experimentación que mostraremos a continuación, se han entrenado con el conjunto de entrenamiento mostrado en la tabla 2.4, es decir un corpus de 128 000 pares de frases, utilizando el esquema de entrenamiento $1^5 2^5 3^5 4^5 5^5$. Con respecto a los modelos de

lenguaje, hemos utilizado modelos de trigramas suavizados mediante la técnica de descuento *Good Turing* y han sido entrenados con las correspondientes 128 000 frases para el para el lenguaje de salida.

Cabe matizar, que en las tablas relativas a los experimentos que expondremos a continuación se incluye:

- El tiempo medio de ejecución en segundos (**secs**) por frase.
- El modelo de error, es decir: porcentaje errores de búsqueda (**ErrBusq**), porcentaje errores del modelo (**ErrModel**) y porcentaje de aciertos (**Aciertos**).
- Por último, las tasas de error (WER y PER), en tanto por ciento.

C.1 Establecimiento de los parámetros de los algoritmos basados en PD.

Para realizar esta experimentación, hemos establecido por defecto las siguientes características básicas del traductor:

- Algoritmo elegido: *IterativeDPSearchM2_opt_W+Z+beam*
- Inicialización heurística: Si
- Parámetros: $W = 12$, $Z = 24$, $L = 4$, y $beam = 5000$
- Número de iteraciones: 3
- Recombinación: No

Influencia del número de traducciones inversas (parámetro W)

Cómo podemos ver en la tabla C.2, a medida que aumenta el valor de W (número de traducciones inversas para cada palabra f de la frase de entrada) aumenta la calidad de la traducción y del mismo modo el tiempo medio por frase. A la vista de los resultados podemos establecer $W = 20$ como valor aceptable, pues a partir de él no se obtienen errores de búsqueda. De cualquier forma no se observan mejoras sustanciales en cuanto a eficacia para valores de W mayores de 10, en cambio y sí aumenta considerablemente el tiempo de ejecución.

Influencia del número de palabras de fertilidad cero (parámetro Z)

Recordamos que el parámetro Z indica el número de palabras de fertilidad cero a utilizar. A la vista de los resultados de la tabla C.3, podemos ver que a partir de un valor $Z = 30$ se obtienen ligeras mejoras en la calidad de la traducción, aunque en todos el modelo de error no cambia. Evidentemente, a mayor valor de Z mayor tiempo de ejecución.

W	secs	ErrBusq	ErrModel	Aciertos	WER	PER
5	92.80	5	80	15	53.27	48.74
10	320.63	5	80	15	52.90	48.11
15	712.80	5	80	15	52.52	47.98
20	1334.30	0	82	18	51.64	46.98
30	3004.21	0	82	18	51.89	46.85

Tabla C.2: Influencia del número de traducciones inversas para una palabra origen dada (parámetro W).

Z	secs	ErrBusq	ErrModel	Aciertos	WER	PER
5	310.54	5	80	15	52.39	47.61
10	351.74	5	80	15	52.39	47.61
20	431.02	5	80	15	52.39	47.73
30	514.20	5	80	15	52.27	47.61
50	699.08	5	80	15	52.02	47.48

Tabla C.3: Influencia del número de palabras de fertilidad cero consideradas (parámetro Z).

Influencia del valor del parámetro *beam*

A la vista de la tabla C.4 podemos ver que a partir de un valor de $beam = 25$ no se obtienen mejoras sustanciales, y sí se deprecian las cualidades el algoritmo.

beam	secs	ErrBusq	ErrModel	Aciertos	WER	PER
5	156.80	6	80	14	53.40	48.49
10	310.83	6	80	14	52.77	48.11
15	399.88	6	80	14	52.77	48.11
25	455.75	5	80	15	52.39	47.73
50	463.86	5	80	15	52.39	47.73

Tabla C.4: Influencia de la poda basada en la búsqueda en haz (parámetro *beam*).

Influencia del rango de longitudes de salida (parámetro L)

Este parámetro, que establece el número de longitudes de salida a testear, tiene mayor impacto en la eficiencia del algoritmo que en su eficacia. De hecho, a la vista de la tabla C.5 podemos ver que un valor de $L = 3$ es más que suficiente para garantizar cierta calidad en la traducción.

L	secs	ErrBusq	ErrModel	Aciertos	WER	PER
1	155.24	9	79	12	57.05	49.50
2	260.17	7	79	14	54.41	48.87
3	360.32	6	80	14	54.03	48.61
4	463.86	5	80	15	52.77	48.24
5	568.97	5	80	15	52.77	48.24
6	671.96	5	80	15	52.77	48.36

Tabla C.5: Influencia del número de longitudes de salida a testear (parámetro L)

Influencia del modelo en la versión por Viterbi

En la tabla C.6 podemos ver los resultados de traducción para la familia de algoritmos *IterativeDPSearchVitMn*. A la vista de estos resultados podemos ver que no se obtienen mejoras con éstas versiones de los algoritmos, al igual que ocurría en el caso de la tarea EUTRANS-I. Esto, es debido principalmente a la gran cantidad de errores del modelo que se obtienen, lo cual nos lleva a pensar en la necesidad de utilizar un conjunto de entrenamiento mucho mayor para obtener modelos de traducción de cierta calidad. De hecho, a mayor complejidad de la tarea a abordar, se necesita mayor cantidad de información (tamaño del corpus) para obtener modelos que proporcionen cierta garantía.

Mod	secs	ErrBusq	ErrModel	Aciertos	WER	PER
1	465.66	6	79	15	55.92	47.86
2	465.48	5	81	14	52.02	47.61
H	485.60	0	87	13	57.18	51.26
3	471.71	5	80	15	53.15	47.10
4	502.82	6	83	11	53.78	48.24
5	650.07	5	84	11	53.65	47.10

Tabla C.6: Influencia del modelo en la versión por Viterbi.

Elección del algoritmo y los parámetros

Vistos los resultados obtenidos en la experimentación llevada a cabo en esta sección, e intentando establecer un compromiso entre eficiencia y aciertos en la traducción, podemos concluir que una buena elección de un traductor será aquel que: utilice el algoritmo *IterativeDPSearchM2*, con inicialización heurística y sin recombinación de hipótesis. En cuanto a los parámetros de optimización,

se establecene los siguientes valores de $W = 30$, $Z = 30$, $It = 3$, $L = 4$, y $beam = 50$.

C.2 Establecimiento de los parámetros de los algoritmos de pila

Establecemos las características básicas del traductor de la siguiente manera:

- Algoritmo elegido: $A^\star$
- Modelo de traducción utilizado: modelo 4
- Parámetros: $A = 5$, $N = 10$, $S = 10000$, $W = 8$ y $Z = 8$
- Heurístico utilizado: TF
- Recombinación: Sí
- Optimización de $addZFert$: Sí, con optimización pospuesta
- Uso de $add2ZFert$: Sí, con optimización heurística

Influencia de número máximo de palabras a alinear por expansión (parámetro A)

En la tabla C.7 podemos ver la influencia del parámetro A con respecto a la eficiencia y aciertos en la traducción, de donde podemos deducir que no existe una clara influencia del efecto de este parámetro. Una vez más, nos aventuramos a decir, que esto es debido al gran número de errores del modelo obtenidos, haciendose patente la necesidad de utilizar un corpus de entrenamiento mucho mayor. De cualquier forma un valor de $A = 4$ parece adecuado.

A	**segs**	**ErrBusq**	**ErrModel**	**Aciertos**	**WER**	**PER**
1	6.67	14	76	10	57.05	52.52
2	58.82	9	79	12	57.18	52.14
3	157.59	10	79	11	57.30	51.64
4	263.44	12	77	11	57.30	52.64
5	351.62	12	77	11	58.94	53.65
6	412.50	12	77	11	61.34	53.40

Tabla C.7: Influencia del número máximo de palabras a alinear por expansión (parámetro A).

Influencia del número de traducciones inversas (parámetro W)

En la tabla C.8 podemos ver los resultados de la experimentación llevada a cabo para distintos valores de este parámetro. Sorprendentemente, para valores mayores de 10 se obtienen más errores de búsqueda y peor calidad en la traducción. Al igual que en el caso anterior atribuimos este efecto a la mala calidad de los modelos utilizados. De esta forma en el resto de experimentos utilizaremos un valor de $w = 8$.

W	segs	ErrBusq	ErrModel	Aciertos	WER	PER
5	76.47	11	78	11	57.05	51.51
10	214.76	10	79	11	57.43	52.14
15	380.96	12	77	11	58.69	54.41
20	541.91	14	75	11	58.94	55.04

Tabla C.8: Influencia del número de traducciones inversas para una palabra origen dada (parámetro W).

Influencia del número de palabras de fertilidad cero (parámetro Z)

El parámetro Z establece el número máximo de traducciones de palabras de fertilidad cero que se consideran al ejecutar el bucle de operaciones $addZFert$, es por tanto un parámetro vital del algoritmo debido a su influencia en la calidad de la traducción y en la eficiencia. En la tabla C.9 se muestran pruebas con distintos valores de este parámetro. Como puede apreciarse, este parámetro sí tiene relevancia en la eficiencia y calidad de los resultados. De hecho, las mejores tasas de error se obtienen para $Z = 30$, y es de esperar que sigan mejorando para valores superiores, aunque esto conlleva unos tiempos de traducción impractibles.

Z	segs	ErrBusq	ErrModel	Aciertos	WER	PER
5	85.60	11	78	11	56.17	50.76
10	217.22	12	77	11	56.55	51.89
20	628.24	8	79	13	56.68	51.89
30	1298.44	8	80	12	54.41	50.50

Tabla C.9: Influencia del número de palabras de fertilidad cero consideradas (parámetro Z).

Influencia del tamaño de las pilas (parámetro S)

Este parámetro limita el número máximo de hipótesis que puede almacenar la pila. Como podemos ver en la tabla C.10, este parámetro, afecta de forma directa a la calidad y eficiencia de la traducción.

S	segs	ErrBusq	ErrModel	Aciertos	WER	PER
1	0.09	62	37	1	81.36	65.24
5	0.45	57	38	5	73.93	62.22
10	1.29	48	45	7	69.65	60.08
50	2.73	33	58	9	65.87	57.56
100	3.95	27	64	9	63.73	56.80
500	15.36	25	65	10	60.33	54.16
700	20.20	18	71	11	58.44	53.53
1000	27.01	17	72	11	58.69	53.53
5000	95.90	11	78	11	56.80	52.14

Tabla C.10: Influencia del tamaño de las pilas (parámetro S).

Podemos ver que valores superiores a $S = 700$ no mejoran los resultados y sí aumenta ligeramente el tiempo de ejecución, con lo que podemos decir que ese sería un valor razonable.

Influencia del número de hipótesis a expandir (parámetro N)

En la tabla C.11 podemos ver el impacto del parámetro N en la eficiencia. Dada la naturaleza de este parámetro, la tasa de aciertos no resulta afectada (recordemos que se trata del número de hipótesis que se expanden en cada iteración, lo cual sólo tiene sentido para algoritmos de múltiples pilas sin *umbral*).

Podemos ver que existe un ligero aumento del coste por frase al aumentar el valor de N, aunque no de forma sistemática. Esto es debido a que hemos utilizado heurísticos en la búsqueda. Si no hubiéramos utilizado heurísticos las diferencias en los tiempos de ejecución serían despreciables, pues en este caso, cada vez que se expande una hipótesis, el resultado de la expansión producirá una serie de hipótesis cuya *puntuación* sería peor que el de las N primeras hipótesis que se seleccionaron para la expansión, ya que las nuevas hipótesis tienen seguramente más palabras origen cubiertas, de manera que el algoritmo estaría funcionando exactamente igual que si N fuera igual a 1.

Elección del algoritmo y los parámetros

Vistos los resultados obtenidos en la experimentación llevada a cabo en esta sección, e intentando establecer un compromiso entre eficiencia y aciertos en la traducción, podemos concluir que una buena elección de un traductor será aquel

N	segs	ErrBusq	ErrModel	Aciertos	WER	PER
5	157.27	10	79	11	57.30	51.64
20	158.14	10	79	11	57.30	51.64
30	157.38	10	79	11	57.30	51.64
40	157.81	10	79	11	57.30	51.64
50	156.56	10	79	11	57.30	51.64
60	156.62	10	79	11	57.30	51.64
70	157.43	10	79	11	57.30	51.64
80	156.65	10	79	11	57.30	51.64
90	155.45	10	79	11	57.30	51.64
100	156.88	10	79	11	57.30	51.64

Tabla C.11: Influencia del número de hipótesis a expandir por iteración (parámetro N).

que: utilice el algoritmo $A^\star$ con el modelo 4, con los siguientes valores de para los parámetros de optimización: $W = 8$, $Z = 8$, $S = 1000$, y heurístico TF.

C.3 Establecimiento de los parámetros de los algoritmos voraces

Para estos algoritmos vamos fijar por defecto las características básicas del traductor de la siguiente manera:

- Algoritmo elegido: *GreedySearch* con *ComputeNeighborhood_opt_S+W+Z*
- Modelo de traducción utilizado: modelo 3
- Parámetros: $W = 12$, $Z = 24$, $S = 9$
- Inicialización: Viterbi

Influencia del número de traducciones inversas (parámetro W)

En la tabla C.12 podemos ver la influencia del parámetro W con respecto a la eficiencia y aciertos en la traducción. A la vista de estos resultados parece claro que este parámetro no es determinante, en cuanto a eficiencia y la calidad en la traducción. También podemos ver que a partir de un valor de 20 no se obtiene prácticamente mejora alguna en la calidad, y sí una ligera depreciación en la eficiencia, por tanto establecer un valor de $W = 20$ sería suficiente.

W	secs	ErrBus	ErrModel	Aciertos	WER	PER
5	4.20	18	75	7	58.19	51.01
10	10.30	18	75	7	58.06	51.01
15	18.39	17	75	8	58.31	51.13
20	28.10	17	75	8	58.31	51.13
25	39.71	17	75	8	58.56	51.39
30	52.69	17	75	8	58.44	51.26
35	67.57	17	75	8	58.69	51.51
40	84.74	17	75	8	58.82	51.26
45	103.18	17	75	8	58.82	51.26
50	123.23	17	75	8	58.82	51.01

Tabla C.12: Influencia del número de traducciones inversas para una palabra origen dada (parámetro W).

Influencia del número de palabras de fertilidad cero (parámetro Z)

El parámetro Z establece el número máximo de traducciones de palabras de fertilidad cero que se consideran al ejecutar el bucle de operaciones $translateAndInsertZFert(\cdot)$, es por tanto un parámetro vital del algoritmo debido a su influencia en la calidad de la traducción y en la eficiencia. En la tabla C.13 se muestran pruebas con distintos valores de este parámetro. Como puede apreciarse, este parámetro tiene un impacto mucho mayor en la eficiencia y en los aciertos que W. Por ejemplo, podemos ver que al pasar de un valor de 5 a un valor de 30, los errores de búsqueda se ven reducidos en un 50%. Por contra, el tiempo medio por frase se ve aumentado en un factor aproximadamente igual a 5. Siguiendo con el criterio de establecer un compromiso entre eficiencia y eficacia, parece razonable establecer este parámetro al valor de $Z = 30$, pues a partir de ahí no se obtienen mejoras sustanciales en cuanto a calidad y sí depreciaciones importantes en el tiempo de ejecución.

A la vista de estos resultados y los anteriores (parámetro W) sería interesante hacer un estudio bidimensional del efecto de estos dos parámetros para ver si realmente el parámetro W afecta o no a las prestaciones de estos algoritmos.

Influencia del tamaño de los segmentos a intercambiar (parámetro S)

El parámetro S establece el número tamaño máximo de segmentos a intercambiar por tanto afecta al bucle de operaciones del tipo $swapSegments(\cdot)$. En la tabla C.14 se muestran los resultados para distintos valores de este parámetro. A la vista de ellos podemos ver que no existe prácticamente efecto de este parámetro en cuanto a la eficiencia, en cambio sí afecta a la calidad de la traducción, pues al pasar de un valor de 1 a un valor de 6, los errores de búsqueda se reducen ligeramente. Del mismo modo también podemos ver que a

Z	secs	ErrBus	ErrModel	Aciertos	WER	PER
5	6.94	17	75	8	58.06	51.13
10	8.45	17	75	8	58.06	51.13
15	10.29	17	75	8	57.93	51.01
20	11.93	17	75	8	58.19	51.26
25	13.46	17	75	8	58.19	51.01
30	15.18	16	76	8	58.06	50.76
35	16.83	15	77	8	58.56	51.13
40	18.49	15	77	8	58.56	51.13
45	20.12	15	77	8	58.56	51.13
50	21.74	15	77	8	58.56	51.13

Tabla C.13: Influencia del número de palabras de fertilidad cero consideradas (parámetro Z).

partir de un valor de 6 se deprecian ligeramente las prestaciones del algoritmo, lo cual nos induce a pensar que el a partir de ese valor el algoritmo cae con mayor facilidad en óptimos locales.

S	secs	ErrBus	ErrModel	Aciertos	WER	PER
1	11.86	17	76	7	57.43	50.88
2	12.79	16	76	8	57.18	50.76
3	12.80	16	76	8	57.18	50.76
4	12.98	16	76	8	57.81	50.88
5	12.92	17	75	8	58.06	51.01
6	13.13	17	75	8	58.06	51.01
7	13.11	17	75	8	58.06	51.01
8	13.08	17	75	8	58.06	51.01
9	13.11	17	75	8	58.06	51.01

Tabla C.14: Influencia del tamaño máximo de los segmentos a intercambiar en la operación $swapSegments(\cdot)$ (parámetro S).

Determinación de la influencia del algoritmo

En la tabla C.15 podemos ver los resultados de la influencia del algoritmo utilizado. En esta tabla $GSoSWZ$ hace referencia al algoritmo *GreedySearch* con *ComputeNeighborhood_opt_S+W+Z* y $GSOpt$ al mismo con *ComputeNeighborhoodOpt*, y los * a sus versiones optimizadas en el cálculo de la *puntuación*. En este experimento hemos utilizado el modelo 3 (de ahí que

se obtengan peores resultados), para poder ver el efecto de la optimización con respecto al cálculo de la *puntuación*.

A la vista de estos resultados podemos ver la depreciación del algoritmo, en cuanto a calidad en la traducción se refiere, con respecto a las optimizaciones de las operaciones, aunque ello suponga una sustancial ganancia en eficiencia. En cambio la mejora en la eficiencia que incluye la optimización del cálculo de la *puntuación* es tremenda, de modo que la ganancia que podríamos obtener con la optimización de las operaciones se ve en gran medida subsanada con dicha optimización. Por tanto, ya podemos adelantar que el estudio e implementación de la optimización del cálculo de la *puntuación* para el resto de los modelos se hace imprescindible para obtener tiempos de ejecución realmente competitivos.

Alg.	**secs**	**ErrBus**	**ErrModel**	**Aciertos**	**WER**	**PER**
$GSoSWZ^*$	3.05	17	75	8	58.06	51.01
$GSoSWZ$	13.11	17	75	8	58.06	51.01
$GSOpt^*$	0.41	19	73	8	58.56	51.39
$GSOpt$	1.71	19	73	8	58.56	51.39

Tabla C.15: Influencia del algoritmo.

Como era de esperar los resultados obtenidos con la optimización del cálculo de la *puntuación* no introducen errores de búsqueda adicionales.

Determinación de la influencia del modelo utilizado

En la tabla C.16 podemos ver los resultados de la influencia del modelo utilizado. En ella se observa que los mejores resultados se obtienen para el modelo 4, aunque sorprendentemente son prácticamente idénticos a los obtenidos con el modelo 2. Es más, el tiempo medio por frase para el modelo 4 es sustancialmente mayor, como era de esperar.

Mod	**secs**	**ErrBus**	**ErrModel**	**Aciertos**	**WER**	**PER**
1	1.31	58	36	6	66.75	57.43
2	7.35	19	73	8	58.19	51.89
H	22.59	16	76	8	62.09	51.51
3	13.11	17	75	8	58.06	51.01
4	41.93	16	71	13	59.32	52.77
5	40.21	15	72	13	59.82	51.76

Tabla C.16: Influencia del modelo.

Elección del algoritmo y los parámetros

Vistos los resultados obtenidos en la experimentación llevada a cabo en esta sección, e intentando establecer un compromiso entre eficiencia y aciertos en la traducción, podemos concluir que una buena elección de un traductor será aquel que: utilice el algoritmo *GreedySearch* básico para el modelo 3, con los siguientes valores de para los parámetros de optimización: $W = 12$, $Z = 30$, y $S = 10$.

C.4 Conclusiones

A la vista de la experimentación llevada a cabo para la tarea HANSARDS podemos concluir que:

- Dada la gran cantidad de errores del modelo obtenidos con todas las aproximaciones, y la complejidad que conlleva esta tarea, hace indispensable la utilización de corpus de entrenamiento mucho mayores para obtener resultados más competitivos.
- Es necesario establecer fuertes restricciones a los parámetros de optimización de los algoritmos, debido a los costes tan elevados que conlleva la traducción de frases de esta tarea.
- Los algoritmos basados en PD obtienen los mejores resultados, debido a la gran limitación que los algoritmos de pila requieren para hacer factible su utilización.
- A diferencia de lo que ocurría con la experimentación llevada a cabo para la tarea EUTRANS-I no existe tanta diferencia entre los resultados obtenidos con los algoritmos voraces que con resto de algoritmos. Esto unido a que los resultados aquí presentados son comparables a los obtenidos con otras aproximaciones, ratifica la alta complejidad de esta tarea para ser tratada de forma automática.