Técnicas de compresión e estruturas de indexación compactas para texto en linguaxe natural e contidos dixitais: aplicacións prácticas

Autor: Eduardo Rodríguez López

Tese de Doutoramento UDC / 2015

Dirixida por:
Ángeles Saavedra Places
Antonio Fariña Martínez

UNIVERSIDADE DA CORUÑA

Técnicas de compresión e estruturas de indexación compactas para texto en linguaxe natural e contidos dixitais: aplicacións prácticas

Autor: Eduardo Rodríguez López

Tese de Doutoramento UDC / 2015

Dirixida por:
Ángeles Saavedra Places
Antonio Fariña Martínez

Departamento de Computación

UNIVERSIDADE DA CORUÑA

Tese doutoral dirixida por
Ángeles Saavedra Places
Departamento de Computación
Facultade de Informática
Universidade da Coruña
15071 A Coruña (España)
Tel: +34 981 167000 ext. 1249
Fax: +34 981 167160
asplaces@udc.es

Antonio Fariña Martínez
Departamento de Computación
Facultade de Informática
Universidade da Coruña
15071 A Coruña (España)
Tel: +34 981 167000 ext. 1352
Fax: +34 981 167160
fari@udc.es

Ángeles Saavedra Places e Antonio Fariña Martínez, como directores, acreditamos que esta tese cumpre os requisitos para optar ao título de doutor e autorizamos o seu depósito e defensa por parte de Eduardo Rodríguez López, de quen se inclúe tamén a sinatura.

A Clara

Resumo

A aplicación dos resultados da investigación ao desenvolvemento software permite crear sistemas innovadores que resolven problemas, organizan procesos ou ofrecen servizos á sociedade dun modo máis eficaz e eficiente.

Na primeira parte desta tese preséntanse dúas novas técnicas da área de investigación en compresión e indexación de texto en linguaxe natural. Dunha banda, un autoíndice de palabras que permite obter unha representación comprimida do texto empregando tan só un 35–40 % do seu tamaño orixinal, á vez que realizar buscas moi eficientes dentro del. Doutra banda, un compresor orientado a frases que permite reducir un texto até ocupar un 25–30 % do seu tamaño orixinal, ofrecendo unha descompresión moi rápida e a posibilidade de realizar buscas eficientes no texto comprimido.

Na segunda parte preséntanse distintas solucións tecnolóxicas que desenvolvemos e integramos en sistemas reais en produción para resolver diferentes aspectos relacionados co consumo de contidos dixitais, como a súa creación e distribución protexida, e como estas fan uso de diferentes resultados da investigación en compresión e indexación para mellorar a eficiencia no almacenamento, acceso, procesado e transmisión dos contidos de natureza textual.

Resumen

La aplicación de los resultados de la investigación al desarrollo software permite crear sistemas innovadores que resuelven problemas, organizan procesos o que ofrecen servicios a la sociedad de una forma más eficaz y eficiente.

En la primera parte de esta tesis se presentan dos nuevas técnicas del área de investigación en compresión e indexación de texto en lenguaje natural. Por una parte, un autoíndice que permite obtener una representación comprimida del texto utilizando tan solo un 35–40 % de su tamaño original, a la vez que realizar búsquedas muy eficientes dentro de él. Por otra parte, un compresor orientado a frases que permite reducir un texto hasta ocupar un 25–30 % de su tamaño original, ofreciendo una descompresión muy rápida y la posibilidad de realizar búsquedas eficientes en el texto comprimido.

En la segunda parte se presentan distintas soluciones tecnológicas que hemos desarrollado e integrado en sistemas reales en producción para resolver diferentes aspectos relacionados con el consumo de contenidos digitales, como su creación y distribución protegida, y cómo estas hacen uso de diferentes resultados de la investigación en compresión e indexación para mejorar la eficiencia en el almacenamiento, acceso, procesado y transmisión de los contenidos de naturaleza textual.

Abstract

The application of research results to software development allows creating innovative systems that solve problems, organize processes or provide services for the society in a more effective and efficient manner.

In the first part of this thesis, we present two new techniques in the research area of compression and indexing of natural language texts. On the one hand, a self-index that obtains a compressed representation of the text using just 35–40 % of its original size, while enabling very efficient searches. On the other hand, a phrase-based compressor that reduces the text to 25–30 % of its original size, providing very fast decompression and the capability of performing efficient searches over the compressed text.

In the second part, we present several technological solutions that we have developed and integrated into real production systems to solve various aspects related to the consumption of digital contents, such as their creation and protected distribution, and how they make use of different techniques that are product of the research in the field of compression and indexing to improve the efficiency of storing, accessing, processing and transmitting textual contents.

Prefacio

Esta tese de doutoramento reflicte a evolución da miña traxectoria investigadora e profesional durante os últimos case 10 anos, e que tentarei resumir de seguido.

Rematado o meu proxecto de fin de carreira de Enxeñería Informática tiven a oportunidade de incorporarme ao Laboratorio de Bases de Datos[1] (LBD) da Universidade da Coruña (UDC) para investigar na área de compresión e indexación de texto en linguaxe natural. Isto deume a posibilidade de entrar a formar parte dun grupo de investigación que xa daquela (2007) comezaba a ter certa relevancia internacional na área de compresión e indexación de texto.

Durante os dous anos seguintes, ademais de completar a miña formación coa realización dos estudos de doutoramento no programa de Computación (Departamento de Computación da UDC) obtendo o Diploma de Estudos Avanzados (DEA), adiqueime fundamentalmente a investigar no deseño e desenvolvemento de novas estruturas compactas e algoritmos para a compresión e indexación de texto en linguaxe natural, sendo os dous resultados máis relevantes os que presento como contribución desta tese. Tamén nesta etapa tiven a posibilidade de participar en varios proxectos de desenvolvemento software que o LBD realizou no marco de diferentes convenios, o que me permitiu aproximarme a outros campos de investigación con actividade no LBD, como as Bibliotecas Dixitais[2] ou os Sistemas de Información Xeográfica[3].

Por aquel entón, Enxenio[4] (*spin-off* da UDC) era unha empresa pequena con apenas catro anos de vida, xurdida da inquedanza de varios membros do LBD por aplicar o seu coñecemento e experiencia ao desenvolvemento de sistemas de

[1] `http://lbd.udc.es`

[2] Convenio de colaboración entre a Asociación de Escritores en Lingua Galega (AELG) e a Universidade da Coruña para a Dixitalización de Fondos para o Centro de Documentación da AELG

[3] Convenio entre a Excma. Deputación Provincial de A Coruña e a Universidade da Coruña para financiar o proxecto da creación da Web Cultura Galega; Convenio administrativo entre a UDC e a Deputación da Coruña para financiar a creación dunha Web de Roteiros Culturais

[4] `http://www.enxenio.es`

información que axudasen ás organizacións a resolver problemas complexos, organizar procesos de negocio ou ofrecer servizos útiles para a sociedade. Esta orixe de Enxenio vencellada a un grupo de investigación como o LBD fixo que, desde o principio, a súa actividade estivese marcada por un alto grado de transferencia tecnolóxica dos resultados de investigación aos seus desenvolvementos software.

A finais da década, o crecemento que Enxenio viña apuntando desde o seu nacemento fixo preciso abordar unha profunda reestruturación do seu modelo organizativo e produtivo, que tivo como consecuencia tamén unha maior demanda de persoal e que supuxo o meu paso á empresa. Entrei a formar parte de Enxenio primeiro como programador, logo con responsabilidades de análise, máis tarde na dirección de proxectos, e finalmente dirixindo a súa liña de contidos dixitais.

Non obstante, o estreito vínculo entre Enxenio e o LBD permitiume manter sempre certo contacto coa investigación e aplicar os resultados obtidos nela a sistemas reais, ben para resolver necesidades concretas dos produtos desenvolvidos, ou ben como proba de concepto en proxectos de carácter máis innovador. É probable que a miña orientación cara a liña de contidos dixitais da empresa non fose casual, senón a evolución lóxica dada a miña formación investigadora, mais nen sequera a día de hoxe, coa perspectiva do tempo, sería quen de aseguralo. Do que non cabe dúbida é de que os contidos dixitais, e máis concretamente a rama que se centra na creación, distribución e visualización de libros electrónicos, representa un marco ideal para poder aplicar nos sistemas desenvolvidos as estruturas de datos e algoritmos de compresión e indexación de texto en linguaxe natural deseñados durante a miña etapa investigadora.

Como consecuencia de todo o anterior, esta tese divídese de forma natural en dúas partes diferenciadas pero, á vez, intimamente ligadas. A primeira parte está centrada na miña etapa investigadora, e nela preséntanse as estruturas e algoritmos para a compresión e indexación de texto resultado da miña actividade en ela. A segunda parte céntrase na miña etapa profesional, e máis concretamente na miña actividade dentro da liña de contidos dixitais de Enxenio, e nela preséntanse distintas solucións tecnolóxicas que desenvolvemos dirixidas ao negocio dos contidos dixitais editoriais, facendo especial fincapé na aplicación a sistemas reais en produción das técnicas de investigación presentadas anteriormente.

Índice de contidos

Índice de figuras

Índice de cadros

Capítulo 1

Introdución

As tecnoloxías da información están presentes en todos os ámbitos das nosas vidas. Desde a aparición e comercialización dos primeiros ordenadores persoais, na década dos 80s do pasado século, até a imparable expansión dos modernos dispositivos móbiles (*smartphones*, *tablets*, *smartwatches* etc.), estas tecnoloxías foron incrementando a súa presenza no noso día a día até acadar un nivel de inmersión inimaxinable anos atrás e unha dependencia delas case absoluta.

Se hai un feito catalizador deste fenómeno este é, sen dúbida, a irrupción de Internet e, máis concretamente, da web (*World Wide Web*) a principios dos 90s. Isto supuxo un pulo decisivo para o desenvolvemento de novas infraestruturas de comunicación dixital e tecnoloxías de fabricación de dispositivos electrónicos conectados á Rede para o consumo de información dixital. Se ben na súa orixe o acceso a Internet estaba restrinxido aos sectores gobernamental, científico e académico, os avances nas técnicas de fabricación de dispositivos electrónicos de consumo (microelectrónica) e a consecuente redución dos custos de acceso ás Tecnoloxías da Información e as Comunicacións (TIC), provocaron o acceso xeneralizado da sociedade á web, sobre todo nos países máis desenvolvidos.

Desde o punto de vista sociolóxico, isto supuxo o inicio dun proceso de democratización no acceso ás TIC, que deu orixe ao que hoxe en día se coñece como a Sociedade da Información. Segundo o portal web Internet Live Stats[1], ao redor do 40 % da poboación mundial ten xa acceso a Internet, cando en 1995 esa porcentaxe era inferior ao 1 %, e o ritmo de crecemento mantense ao redor dun 3 % nos últimos anos, o que fai prever que antes de 2020 máis da metade da poboación mundial disporá de acceso á Rede. Nos países máis desenvolvidos, loxicamente, esas porcentaxes son moito máis elevadas. Así a porcentaxe de penetración de Internet

[1] `http://www.internetlivestats.com/internet-users`

en Europa a finais de 2014 é de máis do 70 %, e en América do Norte está por riba do 85 %.

Este incremento no acceso da sociedade a Internet e ás novas tecnoloxías trae como consecuencia o imparable aumento da cantidade de información dispoñible en formato dixital, da que unha parte importante é de natureza textual que representa linguaxe natural. Xorde así unha necesidade crecente de almacenar, procesar e manexar grandes volumes de información en formato dixital, para o que resultan de gran importancia dous procesos que son a **compresión** e a **indexación** da información.

Podemos definir de forma simplificada a compresión como a aplicación dunha transformación sobre unha información dada para obter unha representación da mesma que utilice unha cantidade de espazo menor. A compresión de información é un campo amplamente investigado xa desde mediados do século pasado, cando comezan a aparecer dentro da área de Teoría da Información diferentes técnicas baseadas en complexas teorías matemáticas que buscan reducir o tamaño das mensaxes a enviar a través das canles de comunicación da época. Na actualidade, o almacenamento e transmisión de información dixital son necesidades transversais a todos os ámbitos da sociedade, o que fai que a compresión de información adquira unha importancia aínda maior. A aplicación de técnicas de compresión da información ten un impacto positivo no seu almacenamento, ao requirir unha cantidade de espazo menor, pero tamén no seu acceso, transmisión e mesmo procesamento, grazas a que existen técnicas que permiten explotar directamente a información comprimida sen necesidade de descomprimila previamente. Por todo o anterior, a compresión de información segue a ser na actualidade un campo de investigación moi activo.

Por outra banda, a indexación refírese ao procesamento da información para construír estruturas auxiliares, chamadas índices, que permitan posteriormente o acceso e recuperación de dita información de forma eficiente. A indexación de información é unha das tarefas básicas da Recuperación de Información (*Information Retrieval*, IR), que é a disciplina que se ocupa da investigación en técnicas para a obtención da información relevante ante unha necesidade ou consulta concreta dun usuario, dentro dunha colección moi grande de recursos de información.

O acercamento de Internet á sociedade abre unha gran oportunidade de negocio aos sectores tradicionais orientados a ofrecer servizos finalistas aos usuarios, que se verán na obriga de abordar, antes ou despois, un proceso de adaptación dos seus modelos de negocio ao novo marco tecnolóxico, acadando aquelas compañías capaces de facelo unha vantaxe competitiva moi importante sobre as que non. A principios dos anos 2000, e favorecidos pola aparición de protocolos de conexión e métodos de pago electrónico seguros, comezaron a xurdir multitude de espazos web que permitían aos usuarios mercar directamente produtos de consumo a través de Internet, no que

supuxo a consolidación definitiva do comercio electrónico (*eCommerce*).

Un dos sectores para os que a popularización do *eCommerce* supón unha oportunidade maior é seguramente o da industria do entretemento. A diferenza de outros sectores que comercializan produtos de natureza material, medios como a televisión, a radio, os videoxogos ou a música poden ser ofrecidos como produtos dixitais (*digital goods*) para o seu consumo directamente a través de Internet. Aínda que en moitos casos isto supón para os provedores de servizos unha reformulación substancial dos seus modelos produtivos e de distribución, os beneficios que ofrece o *eCommerce* son múltiples, como a posibilidade de chegar a un número ilimitado de clientes, ou a redución dos custos de produción e loxística.

Dentro da industria do entretenemento, o sector editorial ocupa unha posición destacada en canto a volume de negocio potencial a través da Rede. Segundo un informe de The Nielsen Company[2], os libros electrónicos ocuparon en 2014 o segundo posto en canto a porcentaxe de intención de compra *online*, só por detrás das entradas a eventos. Esta tendencia, que xa se vén observando desde anos atrás, esperta o interese do sector editorial na incorporación ao negocio da distribución de contidos dixitais a través de plataformas de *eCommerce*. Non obstante, esta transición non vai resultar sinxela para a maioría de compañías do sector, que polo xeral non dispoñen de persoal con coñecementos tecnolóxicos para a produción en formato dixital dos seus catálogos nin da capacidade económica para abordar o desenvolvemento de plataformas *eCommerce* de contidos dixitais propias.

Cabe salientar que a transformación do sector editorial cara ao dixital tamén se veu prexudicada nos últimos anos pola difícil situación económica que desde 2008 estamos a sufrir, e que ten especial incidencia en países como España e rexións como Galicia. Esta crise afectou notablemente ás empresas editoriais, sobre todo ás máis pequenas, que se viron obrigadas a reducir custos para manter os seus negocios, cando non directamente ao peche. Neste contexto, moi poucas editoras tiveron a posibilidade e a determinación de realizar unha aposta decidida polo dixital, e aquelas que o fixeron non obtiveron, agás contadas excepcións, o retorno agardado ás inversións realizadas. Todo isto fai que o sector editorial se atope a día de hoxe nun estado de incerteza, a cabalo entre a convicción de que o futuro está na comercialización dos seus catálogos en formatos dixitais e o escepticismo sobre a viabilidade económica de realizar a transformación dos seus modelos de produción e distribución tradicionais ao novo contexto dixital.

Xorde pois a necesidade no sector editorial de dispor de solucións tecnolóxicas que permitan a creación, venda e distribución de contidos editoriais en formato dixital, que as grandes compañías tecnolóxicas de Internet como Google, Apple ou Amazon, non tardaron en detectar e explotar. Así, desde finais da década dos 2000 comezaron

[2] `http://ir.nielsen.com/files/doc_financials/Nielsen-Global-E-commerce-Report-August-2014.pdf`

a aparecer plataformas tecnolóxicas (as máis destacadas son Google Play[3], Kindle Store[4] de Amazon e iBooks Store[5] de Apple) que ofrecen diferentes alternativas para que as editoriais poidan vender os seus *ebooks*. Aínda que as estratexias de negocio poden variar lixeiramente dunhas plataformas a outras, todas se basean en ofrecer un servizo de venda dos *ebooks* desde a propia plataforma xenérica, que inclúe a xestión de todos os aspectos relacionados co proceso (usuarios, pagos, protección DRM, devolucións, soporte técnico etc.) a cambio dunha xenerosa comisión por venda, que normalmente está ao redor do 30 %.

Son varias as vantaxes que ofrece para as editoras o emprego destas plataformas xenéricas. Quizais a principal sexa a de poder poñer os libros dixitais á venda de forma case inmediata, sen necesidade de afrontar o desenvolvemento dunha custosa plataforma de distribución dixital propia. Pero tamén non ter que preocuparse da xestión dos usuarios, da seguridade nos pagos electrónicos, da protección dos contidos distribuídos ou da asistencia técnica aos clientes.

Un dos principais reclamos que ofrecen estas plataformas para as editoras é a visibilidade que poden alcanzar os *ebooks* publicados, dada a gran cantidade de usuarios cos que contan, aínda que en realidade trátase de usuarios non segmentados, consecuencia da diversidade de servizos e produtos que ofrecen estas plataformas non específicas de libros dixitais, e cun ámbito xeográfico e cultural practicamente ilimitado. A falta de control sobre os compradores, que son clientes das plataformas e non das editoras, é precisamente unha das principais desvantaxes do emprego destas plataformas de terceiros, ao non permitir, por exemplo, que a editora poida levar a cabo campañas de seguimento, promoción e fidelización dos seus clientes. Por outra parte, a competencia dentro delas é moi alta, o que fai moi difícil que os libros dunha editora alcancen a relevancia agardada por ela. Esta elevada competencia ten o seu impacto tamén nas políticas de prezos, pois as pequenas editoras están a competir con outras grandes que, por economía de escala, poden ofrecer os seus libros dixitais a prezos cos que as pequenas editoras non poden competir.

O informe *El sector del libro en España*[6], publicado polo Ministerio de Educación, Cultura e Deporte en abril de 2015, ofrece unha serie de datos reveladores de cal é a realidade do sector editorial a día de hoxe en España. Con respecto do modelo produtivo, destacar o feito de que en 2014 cae por segundo ano consecutivo a edición de libros dixitais (un 1,9 % con respecto a 2013), mentres que a facturación por vendas de *ebooks* segue a súa tendencia alcista (un 8,1 % en 2013, sen datos aínda de 2014). Tamén decrece (un 4 %) o número de editoriais que publican en dixital, que representan só o 21,9 % das totais. Estes datos semellan indicar unha tendencia

[3] `https://play.google.com/store/books`

[4] `http://www.amazon.es/ebooks-kindle/b?node=827231031`

[5] `https://itunes.apple.com/es/genre/libros/id38`

[6] `http://www.mecd.gob.es/cultura-mecd/dms/mecd/cultura-mecd/areas-cultura/libro/mc/observatoriolect/redirige/estudios-e-informes/elaborados-por-el-observatoriolect/sector-libro-abril2015.pdf`

cara un mercado editorial en dixital cun alto grado de concentración, que xa se pode albiscar no feito de que, en 2014, case o 30 % da edición dixital está en mans de apenas 10 editoras. En canto a formatos, cabe sinalar que por primeira vez en 2014 o número de *ebooks* editados en EPUB[7], un formato específico para publicacións dixitais, supera aos editados en PDF, que é un formato máis orientado a impresión, sendo estes os dous formatos que representan ao redor do 80 % da produción de libros dixitais. Este dato é moi significativo, pois reflicte o resultado do proceso de especialización en edición dixital que se está a producir no sector nos últimos anos. Os programas de edición empregados habitualmente para a elaboración dos libros en papel, como Adobe InDesign ou QuarkXPress, permiten exportar de forma inmediata o *ebook* en formato PDF, e por este motivo a maioría dos *ebooks* que se publicaban até o de agora eran basicamente a maqueta de impresión do libro en formato PDF, no mellor dos casos, con algunhas modificacións como a eliminación das marcas de corte para imprenta ou a unificación do sangrado nas páxinas pares e impares. Mais o formato PDF caracterízase polo emprego dun tamaño de páxina fixo (*fixed layout*) que non resulta axeitado para a súa visualización en dispositivos de pantalla reducida, como os *smartphones*, polo que non tardaron en aparecer novos formatos de publicación dixital (*reflowable formats*), como EPUB ou MOBI (Mobipocket[8]), nos que o contido se axusta ao tamaño de pantalla para ofrecer unha correcta visualización en dispositivos con características físicas moi diferentes, como a un ordenador persoal, un *eReader* ou un *smartphone*. Desafortunadamente para as editoras, a edición de *ebooks* nestes formatos que os usuarios demandan na actualidade require de maiores coñecementos técnicos (están baseados en estándares web como XHTML, CSS ou SVG), e aínda que os programas de maquetación foron incorporando funcionalidades de exportación a eles o resultado que se obtén é na maioría dos casos un libro dixital de escasa calidade. En realidade, o problema vai máis aló do simple feito de coñecer e manexar unha serie de estándares e tecnoloxías que permitan empaquetar un contido en formato EPUB ou MOBI. Implica un cambio substancial na metodoloxía de traballo e nas decisións a tomar no deseño da publicación dixital. É erróneo tratar de realizar a transformación automática de libros maquetados a *ebooks*, pois o enfoque de partida debe ser diferente, e as decisións de edición non deben estar condicionadas por un tamaño de páxina fixo, senón que deben seguir unha serie de recomendacións e boas prácticas que permitan garantir a accesibilidade dos contidos en dispositivos con características físicas e de usabilidade moi diferentes.

No mesmo informe pode observarse como un 64,5 % da facturación do libro dixital en 2013 procede da venda en plataformas comerciais xenéricas de terceiros, que representan a principal canle de distribución de *ebooks*, aínda que un 10 % da facturación procede xa de plataformas propias dunha editora concreta ou conxuntas de varias editoras, e máis dun 20 % da venda directa desde as propias páxinas web

[7] `http://idpf.org/epub`

[8] `http://www.mobipocket.com/dev/article.asp?BaseFolder=prcgen&File=mobiformat.htm`

destas. En total, máis dun 30 % dos ingresos por vendas de *ebooks* proceden por tanto de canles de distribución resultantes de iniciativas particulares ou conxuntas, que buscan alternativas aos condicionantes que supón o emprego de plataformas comerciais de terceiros, fundamentalmente, as altas comisións por venda e a elevada competencia existente en elas.

Por último, o citado informe tamén revela unha aposta por novos modelos de distribución, máis acordes cos tempos que corren, e diferentes ao da venda individual de libros (herdanza dos modelos tradicionais). Desde 2013, percíbese unha clara aposta pola oferta dos catálogos editoriais en *streaming* (lectura *online*), o que representa xa un 48,6 % da facturación, e por modelos de subscrición xa implantados con éxito noutros sectores da industria do entretemento, como o das plataformas de vídeo por Internet, tipo NetFlix, ou de música, tipo Spotify.

Todo isto fai que o sector editorial se atope na actualidade nun estado de indefinición e tamén de certo escepticismo sobre as oportunidades de negocio que ofrece a publicación dixital, con especial incidencia nas pequenas e medianas editoras, ao seren as que se atopan con maiores dificultades económicas e de recursos humanos para dar solución aos problemas da elaboración e distribución de contidos dixitais.

Coñecedores de toda esta problemática, froito de varios anos de experiencia no deseño e desenvolvemento de sistemas a medida para diferentes editoras, decidimos iniciar a principios da década de 2010 un proxecto estratéxico para ofrecer unha solución integral ás empresas do sector editorial, a través do desenvolvemento de produtos e solucións tecnolóxicas accesibles que permitan levar a cabo as diferentes tarefas do negocio da publicación dixital, desde a creación dos contidos até a súa distribución e consumo. As sinerxias existentes co Laboratorio de Bases de Datos da UDC, un grupo de investigación punteiro en compresión e indexación de información textual (na base de calquera publicación dixital), permitiunos ademais a incorporación aos sistemas desenvolvidos de solucións tecnolóxicas de moi baixo nivel (estruturas de datos compactas e algoritmos) que aportan un alto grado de innovación a estas ferramentas.

Contribucións desta tese

Na primeira parte desta tese presentamos dúas técnicas que ofrecen resultados relevantes na área de compresión e indexación de texto en linguaxe natural. Dunha banda, un autoíndice orientado a palabras que permite obter unha representación comprimida do texto empregando unha cantidade de espazo comparable á dos compresores de texto do estado da cuestión (35–40 % do tamaño orixinal), mais que ao mesmo tempo ofrece a posibilidade de realizar buscas sobre o texto comprimido, superando a eficiencia de estruturas de busca clásica como os índices invertidos na busca de palabras e sobre todo frases. Presentamos ademais dúas variantes do

autoíndice, o orixinal `WCSA` (Word-Based CSA) e o `FWCSA` (Flexible Word-Based CSA) que incorpora unha capa de presentación que permite ampliar as capacidades de busca do autoíndice con outras operacións útiles en Recuperación de Información, como a busca por raíces de palabras ou sen diferenciar maiúsculas e minúsculas, mantendo a eficiencia espacial e temporal do autoíndice orixinal. As ideas principais destes autoíndices foron publicadas en *Proceedings of the 15th International Symposium on String Processing and Information Retrieval (SPIRE 2008)* [BFN+08], e ampliadas posteriormente nun artigo da revista JCR *ACM Transactions on Information Systems (TOIS), 2012* [FBN+12], citado en 22 ocasións (segundo Google Scholar).

E doutra banda, presentamos unha nova técnica de compresión de texto en linguaxe natural, á que denominamos `V2VDC`, baseada na codificación estatística con `ETDC` de frases[9] repetitivas, seleccionadas mediante o emprego de diferentes heurísticas a partir da información do array LCP (*Longest Common Prefix*). Esta nova técnica permite obter excelentes porcentaxes de compresión (por debaixo do 25 % do tamaño do texto orixinal), á altura das mellores técnicas do estado da cuestión en compresión de texto. Ademais ofrece unha descompresión moi rápida e con acceso aleatorio, e a posibilidade de buscar eficientemente sobre o texto comprimido, máis rápido que calquera outra técnica de compresión que admita buscas no texto comprimido. Esta técnica foi publicada en *Proceedings of the 2010 Data Compression Conference (DCC 2010)* [BFL+10].

Na segunda parte da tese presentamos distintas solucións dirixidas ao sector editorial para resolver a problemática que xorde ao redor da creación e distribución de contidos dixitais, nas que facemos uso de técnicas resultado da investigación en compresión e indexación de texto en linguaxe natural para mellorar o rendemento das operacións de almacenamento, acceso, procesado e transmisión dos contidos textuais (melloras que derivan da eficiencia espacial e temporal destas técnicas). Tamén explicamos como estas solucións foron integradas en distintas plataformas reais en produción dotándoas dun alto grado de innovación.

Para a distribución de *ebooks* a través de Internet, clasificamos primeiramente as solucións en aquelas que permiten a descarga dos contidos e o seu consumo posterior sen conexión, e as que só permiten o consumo *online* mediante *streaming*. Para a modalidade de descarga presentamos dúas alternativas para a protección dos contidos dixitais que son, un DRM forte independente do formato de publicación que garante que só os usuarios autorizados acceden aos contidos, e un DRM lixeiro, menos invasor para o usuario, pero que resulta suficiente en determinados contextos onde o acceso aos contidos non precisa ser tan restritivo. Para a distribución en *streaming* presentamos unha solución accesible que permite a lectura de *ebooks* desde practicamente calquera dispositivo a través dunha aplicación desenvolvida con tecnoloxía HTML5 (Xcloud Reader), que non precisa dispor de conexión permanente a Internet. Esta solución intégrase nun produto comercial, Xcloud Bookstore, unha

[9] Considérase que unha palabra é unha frase de lonxitude 1.

plataforma dirixida a editoras para a comercialización do seu catálogo de *ebooks* en formato EPUB. Tamén para a distribución mediante *streaming*, presentamos unha alternativa para a transmisión comprimida dos contidos textuais, que resulta de gran utilidade en contextos con ancho de banda limitado ou nos que é conveniente reducir o volume de datos transmitidos.

Por último, no que á creación de contidos dixitais se refire, presentamos E-ditor, unha ferramenta SaaS (*Software as a Service*) dirixida principalmente a editoras que permite crear *ebooks* de distinta natureza e en distintos formatos de publicación desde unha interface de traballo única. Esta ferramenta aborda o proceso de creación de *ebooks* desde unha perspectiva innovadora consistente na separación dos diferentes aspectos que forman parte da publicación dixital, o que permite separar en fases o proceso creativo e establecer un *workflow* que guía ao usuario durante o proceso de creación. E-ditor conta cun rexistro de software, e está dispoñible publicamente desde 2013 en `http://www.e-ditor.es`, contando na actualidade con case 900 usuarios activos. As principais características desta ferramenta foron publicadas en *Proceedings of the 5th International Conference on Computer Supported Education (CSEDU 2013)* [RLPC+13] e tamén como capítulo do libro *Visibilidad y divulgación de la investigación desde las Humanidades Digitales. Experiencias y proyectos*, Baraibar, Álvaro (ed.)

Estrutura da tese

O resto da tese está organizada en dúas partes básicas, máis unha final coas conclusións e retos futuros.

A primeira parte leva por título **Compresión e indexación de texto** e está centrada na miña actividade investigadora na área de estruturas avanzadas e algoritmos de compresión, indexación e recuperación de textos. No capítulo "2.Estado da cuestión" faise unha introdución ás áreas de compresión e indexación de texto e ás técnicas máis relevantes en cada unha delas. O capítulo "3.Conceptos previos" inclúe unha definición en maior detalle das estruturas de datos e algoritmos que se empregan posteriormente nas técnicas que se presentan como contribucións da tese. No capítulo "4.Autoíndices orientados a palabras" preséntase unha das contribucións da tese, dúas variantes de autoíndices orientadas a palabras (`WCSA` e `FWCSA`) para texto en linguaxe natural que permiten reducir até un 30 % o seu tamaño á vez que resolver de forma eficiente as buscas de palabras e frases. O capítulo "5.Compresión orientada a frases" presenta unha técnica de compresión de texto en linguaxe natural (`V2VDC`) que emprega como símbolos de entrada frases de lonxitude variable que se codifican estatisticamente con códigos de lonxitude tamén variable (`ETDC`), e que supón outra contribución desta tese.

A segunda parte da tese, **Contidos dixitais: aplicacións prácticas**, está

enfocada na miña actividade no desenvolvemento de solucións software dentro da liña de contidos dixitais de Enxenio. O capítulo "6.Panorámica dos contidos dixitais" ofrece unha visión da situación actual do negocio dos contidos dixitais dirixidos ao consumo, centrada sobre todo nos contidos editoriais. O capítulo "7.Distribución de contidos dixitais" presenta, como outra das contribucións da tese, diferentes solucións tecnolóxicas desenvolvidas para a distribución de contidos dixitais a través de Internet, e como estas se integraron en distintas plataformas comerciais en produción. O capítulo "8.Creación de contidos" presenta E-ditor, unha ferramenta SaaS para a creación de contidos dixitais multiformato que define un *workflow* que guía o proceso de edición dixital, e que se inclúe como última contribución da tese.

Finalmente, en **Conclusións e retos futuros** inclúense os capítulos "9.Conclusións", no que se fai un breve resumo das contribucións e conclusións obtidas no marco da tese, e "10.Retos futuros", no que expoñen as liñas de traballo futuro relacionadas coas contribucións presentadas.

Parte I

Compresión e indexación de texto

Capítulo 2

Estado da cuestión

A revolución tecnolóxica que vivimos desde fai xa máis de dúas décadas está a transformar todos os ámbitos da sociedade, e dou orixe ao que hoxe en día se coñece como a Sociedade da Información. Internet e as novas tecnoloxías da información promoveron a aparición de grandes coleccións de datos en formato dixital, como a dispoñible en espazos web, redes sociais, bibliotecas dixitais, bases de datos documentais, coleccións biolóxicas etc. Para poder almacenar e acceder de forma eficiente a toda esta información é preciso dispor de técnicas que permitan reducir o espazo ocupado pola información (compresión) e que proporcionen capacidades de busca (indexación) dentro dela.

A Recuperación da Información é a área que se ocupa da investigación en técnicas que permiten obter dentro dunha colección grande de datos, aqueles que resultan relevantes para resolver unha necesidade de información concreta. Unha das ramas dentro da Recuperación da Información é a da Recuperación de Texto, que é a que traballa con datos de natureza textual. A investigación presentada no ámbito desta tese está centrada nesta rama e, máis concretamente, no estudo e desenvolvemento de estruturas de datos e algoritmos para comprimir e indexar texto en linguaxe natural.

Dentro das técnicas clásicas de indexación de texto en linguaxe natural destacan, polo estendido do seu emprego, os índices invertidos [WMB99, BYRN99]. Os índices invertidos son estruturas que se constrúen a través do preprocesamento dos textos, e que ofrecen unha boa eficiencia para realizar buscas de palabras dentro de grandes coleccións de texto en linguaxe natural, o que fai que se atopen a día de hoxe na base da maior parte dos sistemas de recuperación de información textual, como os motores de busca na web ou os buscadores integrados en bibliotecas dixitais e outros repositorios documentais. Podemos clasificar os índices invertidos en dous

tipos segundo os resultados que permiten obter nas operacións de busca, e que son:

- Orientados a documentos. Nesta modalidade de índices invertidos os elementos de referencia da indexación, e que se ofrecen como resultado das procuras, son os documentos (*document retrieval*). Os índices invertidos orientados a documentos van permitir recuperar aqueles documentos que son relevantes para unha consulta específica, normalmente que conteñen un ou varios termos de busca. Os índices invertidos orientados a documentos poden seguir unha aproximación *booleana*, na que o resultado é simplemente o subconxunto de documentos relevantes, ou baseada en *ranking*, na que o resultado é unha lista dos documentos ordenados segundo a súa relevancia para a consulta formulada.
- Orientados a palabras. Nesta modalidade dos índices invertidos almacénanse as posicións exactas nas que aparece cada palabra dentro do texto, o que vai permitir recuperar a localización exacta dos termos buscados dentro da colección, a diferenza dos orientados a documentos que non permiten determinar en que posicións exactas aparecen os termos de busca dentro deles.

Nesta tese imos centrarnos nas técnicas que permiten recuperar as posicións exactas onde aparece un determinado patrón de busca dentro do texto, e consecuentemente, por exemplo, ofrecer ao usuario como resultado dunha procura un fragmento de texto ao redor do patrón buscado, aínda que tamén veremos na sección 7.1.1 como é posible combinar un índice invertido a documentos con unha técnica orientada a palabras para desenvolver un motor de busca que permita obter as posicións dos termos buscados dentro dun catálogo completo de *ebooks*.

Un índice invertido orientado a palabras consiste basicamente nun conxunto dos termos (*vocabulario*) que aparecen no texto indexado e, para cada un dos termos, unha lista invertida (*posting list*) coas posicións, en orden ascendente, nas que o termo aparece dentro do texto. Esta estrutura permite responder rapidamente a buscas de palabras, pois sería simplemente atopar a palabra no vocabulario e devolver a súa lista invertida. Tamén permite resolver consultas de frases mediante a intersección de listas, isto é, obter a lista invertida de cada palabra por separado e logo comprobar se esas palabras van consecutivas, formando a frase buscada, nalgunha das súas aparicións. A busca de frases pode resolverse de forma eficiente grazas a que as listas invertidas están ordenadas, mais existen distintos métodos para realizar a intersección de listas, que segue a ser unha área de investigación moi activa na actualidade [DM00, BY04, BK02, BLOL06, BLLS09, LBK15].

Desafortunadamente, os índices invertidos orientados a palabras presentan o inconveniente de precisar moito espazo xa que, aparte de ser preciso manter o texto, unha representación plana dun índice invertido pode ocupar entre un 40 % e un 80 % do que ocupa o propio texto. En principio podería parecer que o espazo de almacenamento dos índices non debería ser un problema, debido á ampla

dispoñibilidade que existe na actualidade de dispositivos de almacenamento masivo, cada vez cun custo menor. Mais o problema real non é tanto a cantidade de espazo que se precisa para almacenar o índice, senón a necesidade de proporcionar un acceso eficiente a el. Para lograr isto, é conveniente aloxar os índices nos niveis máis altos da xerarquía de memoria (rexistros da CPU, cachés e memoria principal), que son os que ofrecen un acceso considerablemente máis rápido que, por exemplo, o acceso a disco. O problema está en que eses niveis superiores son os que ofrecen tamén unha maior limitación de espazo, e de aquí xorde a necesidade de empregar técnicas de compresión que permitan reducir o tamaño dos índices.

Como un sistema de busca que utilice índices invertidos vai precisar, ademais dos propios índices, manter o texto orixinal para resolver as buscas, empregaranse técnicas de compresión tanto para reducir o espazo ocupado polo texto como o do propio índice. Como veremos de seguido, existen técnicas de compresión que permiten reducir o tamaño dos índices invertidos a preto dun 20 %–30 % do seu espazo orixinal e o texto a un 25 %–30 % tamén do seu tamaño inicial, o que fai que a día de hoxe sexa posible ter unha colección comprimida e indexada nun espazo ao redor do 60 % do tamaño orixinal da colección. A continuación preséntanse as principais técnicas que se empregan para comprimir os índices invertidos e o texto.

- Representacións comprimidas de listas invertidas. A compresión dos índices invertidos baséase habitualmente en aproveitar o feito de que as listas de posicións están ordenadas para, en lugar de almacenar os valores absolutos, gardar os incrementos entre posicións consecutivas na lista, reducindo así a dimensión dos valores a almacenar. Esta representación incremental das listas invertidas codifícase posteriormente con algunha técnica de codificación de enteiros que favoreza a compresión de valores pequenos [WMB99, ZM06] como os códigos δ, os códigos γ, ou os códigos *Rice*.

  A compresión de listas invertidas baseada no emprego destas técnicas de codificación de enteiros permite obter porcentaxes de compresión moi boas, mais o acceso aos valores das posicións é lento debido á súa ineficiencia na descompresión. Isto, unido ao incremento da capacidade das memorias RAM de hoxe en día, fai que cobre importancia o emprego de técnicas de compresión das listas invertidas que, se ben non obteñen os niveis de compresión dos códigos *Rice*, si ofrecen unha maior velocidade na descompresión. Xurdiron así os *byte codes* [WZ99, CM10] que xeran códigos aliñados a byte (os códigos son secuencias de bytes en lugar de bits), e máis recentemente técnicas que manexan códigos aliñados a palabras (enteiros) como a familia *Simple-X*, entre os que cabe sinalar a técnica *Simple-9* [AM05, AM10]. *Simple-9* é capaz de empaquetar varios códigos asociados ás diferenzas das listas invertidas dentro dun único enteiro, o que permite que o proceso de descodificación traballe a nivel de enteiro e sexa deste modo máis eficiente. Na mesma liña créase a representación *PForDelta* que traballa a nivel de bloques de valores (habitualmente 128) que

son codificados de forma que se favoreza a súa compresión [Hem05, ZHNB06]. Seguindo nesta liña, nos últimos anos desenvolvéronse múltiples algoritmos de codificación de enteiros que usan instrucións SIMD (Simple Instruction Multiple Data) dispoñibles nos procesadores modernos para axilizar o proceso de descodificación [AM10, SGL10, SGR+11, Tro14, LBK15]. Neste caso a intersección de listas faise normalmente descodificando as listas nun primeiro paso e a seguir aplícase algún dos algoritmos de intersección do estado da cuestión.

Por outra banda, e sen perder de vista o feito de que algúns dos algoritmos de intersección máis rápidos requiren de acceso directo ás listas invertidas, outra liña de investigación é a creación de representacións comprimidas de listas invertidas que usan *sampling* para facilitar o acceso directo á representación comprimida das mesmas. Entre as representacións máis destacables nesta liña cabe citar os traballos de Culpepper e Moffat [CM07, MC07, CM10], Sanders e Transier [ST07, ST08, TS10], e máis recentemente o traballo de Ottaviano e Venturini [OV14].

- Representacións comprimidas de texto para bases de datos textuais. A compresión de textos é unha técnica que permite reducir o tamaño dun texto e consecuentemente os recursos necesarios para almacenalo en disco, ou para transmitilo a través dunha rede. Existen dúas principais familias de compresores: os estatísticos e os baseados en dicionario. Estes últimos non posúen propiedades desexables para o seu emprego con bases de datos textuais, pero si son interesantes cando se pretende almacenar unha colección, sen agardar capacidade de busca sobre o texto comprimido, e tamén se o que se pretende é a transmisión de datos (sen requirimentos de tempo real).

  Os compresores estatísticos, dos que o seu expoñente máis representativo é a codificación Huffman [Huf52], son os máis adecuados para a compresión de bases de datos textuais. Estes compresores procesan o texto a comprimir para crear un modelo que permita detectar os símbolos existentes e a súa frecuencia, e usan dito modelo para asociar códigos máis curtos aos símbolos máis frecuentes, conseguindo así a compresión. Habitualmente os compresores que usan Huffman son baseados en caracteres (os caracteres son os símbolos a comprimir) e os códigos xerados son orientados a bit (os códigos son secuencias de bits de lonxitude variable). Cando traballan sobre texto en linguaxe natural, a compresión obtida é tradicionalmente pobre (en torno ao 60 %).

  A finais da década dos 80 xorde unha idea que viría a revolucionar a compresión de textos en linguaxe natural, e que consiste en empregar palabras en lugar de caracteres como símbolos a comprimir [BSTW86]. Isto tivo dous efectos principais. Dunha banda a compresión obtida ao usar codificación Huffman binaria sobre as palabras do vocabulario [Mof89] permite comprimir un texto até alcanzar un 25 %–30 % do tamaño orixinal, fronte ao 60 % da codificación

Huffman orientada a caracteres (a distribución de frecuencias das palabras é moito máis sesgada). E doutra banda, o feito de utilizar palabras fai que tanto os índices invertidos como o compresor/descompresor do texto usen o mesmo vocabulario, o que vai permitir realizar indexación directamente sobre o texto comprimido, apuntando a posicións dentro do texto comprimido en lugar de facelo ás posicións sobre o texto sen comprimir. Deste xeito, a colección pode manterse comprimida todo o tempo, e só é necesario realizar descompresión (parcial) para presentarlle un documento ao usuario. Incluso é posible realizar *string matching* directamente sobre o texto comprimido sen máis que comprimir o patrón de busca e despois buscalo directamente dentro do texto comprimido.

A finais dos 90 [MNZB98, MNZBY00] xorden dúas técnicas denominadas *Plain Huffman* e *Tagged Huffman* que, como o seu nome indica, usan codificación Huffman, pero neste caso D-aria (bytes) en lugar de binaria. Usar códigos orientados a byte fai que a compresión se degrade ata un 30 %–35 %, pero a cambio a descodificación é moito máis rápida. No caso particular do *Tagged Huffman* a compresión empeora aínda uns 4–5 puntos porcentuais máis que no *Plain Huffman* debido a que se emprega o primeiro bit de cada byte como unha marca (*tag*) para indicar se o byte é o primeiro dun código ou non. O feito de usar unha marca fai que os códigos xerados polo *Tagged Huffman* teñan unha propiedade moi interesante: son autosincronizados e libres de sufixo (un código non é sufixo doutro máis longo), e tamén libres de prefixo por usar codificación Huffman[1]. Isto permite, por unha banda, que sexa posible acceder directamente a calquera posición do texto comprimido de polo tanto a descompresión aleatoria, e por outra, realizar buscas directas sobre o texto comprimido até 8 veces máis rápido [MNZBY00] que sobre o texto sen comprimir mediante o emprego de algoritmos de *string matching* baseados en Boyer-Moore [BM77, Hor80].

As técnicas de compresión que permiten buscas directas eficientes e descompresión aleatoria desde calquera posición do texto comprimido abriron a porta á creación dos chamados índices invertidos a bloques [NMN+00]. Estes índices non gardan todas as posicións onde aparece cada palabra na colección, senón que simplemente gardan en que bloques aparecen. O tamaño de bloque é un parámetro que se introduce no proceso de indexación que permite axustar a granularidade do índice invertido: un tamaño de bloque grande (por exemplo 256KB) dá lugar a índices invertidos que ocupan pouco espazo (2 %–3 % do tamaño da colección), mentres que tamaños de bloque pequenos (por exemplo 512 bytes) xera índices máis pesados (en torno ao 10 %), pero a cambio fainos

[1] A codificación Huffman é coñecida por ser a codificación libre de prefixo óptima. Un código libre de prefixo é interesante pois permite unha rápida descodificación ao non precisar mirar os códigos que veñen a continuación (*look-ahead*); é dicir, no momento en que se le o último símbolo do código xa é posible determinar a que símbolo do vocabulario fai referencia.

moito máis áxiles en tempo de busca.

Ao non dispor de información das posicións nas que aparecen as palabras dentro dos textos, tanto os índices invertidos por bloques como os que apuntan a documentos poden ser empregados en tempo de busca para filtrar en que bloques/documentos aparece un termo, pero a continuación vai ser preciso acceder aos bloques/documentos candidatos (que estarán comprimidos) e aplicar *string matching* para localizar as posicións exactas onde aparece dito termo. No caso da busca de frases, só esta fase de escaneo do bloque/documento permite confirmar se as palabras que compoñen a frase aparecen en posicións adxacentes, pois o índice invertido só indica que ambas as dúas aparecen no bloque/documento candidato.

As interesantes propiedades do *Tagged Huffman*, a saber *i)* boa compresión, *ii)* autosincronizado (acceso directo e descompresión aleatoria), e *iii)* realización de buscas directas eficientes sobre o texto comprimido, convertérono na técnica de referencia no eido das bases de datos textuais até que en 2003 [BINP03, BFNP07] aparece o *End-Tagged Dense Code* que, mantendo as súas cualidades, evita a necesidade de empregar códigos Huffman para obter unha codificación libre de prefixo. Esta nova técnica utiliza tamén un bit de marca, pero para indicar cando un byte é o derradeiro byte dun código. En vez de Huffman, emprega unha codificación *densa* que non desaproveita ningunha das 2^7 combinacións dos 7 bits restantes de cada byte, e que á vez resulta máis simple dado que a codificación é totalmente secuencial e non precisa da construción dunha árbore, como no caso de Huffman. O resultado é unha técnica que comprime mellor que o *Tagged Huffman* (uns 3 puntos porcentuais) e dista da codificación óptima, o *Plain Huffman*, en menos dun punto porcentual. Ademais, os procesos de codificación, descodificación e busca son os máis rápidos do estado da cuestión. Na sección 3.1 faremos unha explicación en maior profundidade desta técnica, que posteriormente se emprega nunha das contribucións desta tese, o `V2VDC` (capítulo 5).

En definitiva, acabamos de ver que na actualidade é posible obter unha solución de busca con índices invertidos que, dependendo da granularidade elixida, pode precisar unha cantidade de espazo entre o 35 % e o 60 % do tamaño do texto orixinal.

Por outra banda, os compresores orientados a byte que vimos anteriormente están baseados en palabras e usan códigos libres de prefixo, e como tamén comentamos obteñen porcentaxes de compresión[2] aceptables, aínda que en ningún caso poderán mellorar os niveis de compresión obtidos polo *Plain Huffman* [Sha01]. O motivo é que usan o que se chama unha codificación de orde cero (asúmese que a frecuencia relativa dunha palabra é independente das demais palabras da colección), e é coñecido que a

[2] Entendemos por porcentaxe de compresión $= 100 \times TC/TP$. onde TC e TP refírense ao tamaño do texto comprimido e ao do texto plano orixinal respectivamente.

compresión obtida polo óptimo (*Plain Huffman*) que denominamos L_H está acotada inferior e superiormente pola expresión:

$$H_D \leq L_H < H_D + 1,$$

onde $H_D = \sum_{0 \leq i < n} p_i \log_D \frac{1}{p_i}$ é o que se coñece como a *entropía de orde cero* da distribución de frecuencias usada pola codificación. Isto tradúcese en que en coleccións típicas de textos en inglés non podería ser posible baixar do 30 % en porcentaxe de compresión. Nesta tese, presentamos como contribución unha nova técnica de compresión denominada *Variable-To-Variable Dense Code* [BFL+10] que permite superar este límite e acercarse a niveis de compresión máis próximos aos que ofrecen técnicas máis potentes como o *P7zip*[3] ou *PPMd*[4], que comprimen moito pero son moi lentas en compresión e descompresión e non permiten buscas. Esta técnica, que será explicada en detalle no capítulo 5, obtén porcentaxes de compresión en torno ao 23 %–25 %, e, aínda que é relativamente lenta en compresión, permite procesos de descompresión e busca tan eficientes como os do *End-Tagged Dense Code*, e incluso mellores, o que a sitúa entre os descompresores máis rápidos, e que ademais permiten buscar sobre o texto comprimido, do estado da cuestión. A chave da súa eficiencia está en que comprime non só a nivel de palabras, senón frases de lonxitude arbitraria, sobre as que aplica unha codificación densa. Deste xeito, é posible que unha frase longa (con moitas palabras) que apareza varias veces, sexa codificada cun único código, en lugar de codificar as palabras por separado.

Como vimos ao inicio desta sección, a compresión é interesante non só no ámbito das bases de datos textuais, senón que tamén o é cando cómpre realizar transmisión de información a través dunha rede. As técnicas estatísticas vistas con anterioridade non son axeitadas para este fin xa que non admiten compresión en tempo real ao utilizar un modelado semiestático (ou de dúas pasadas): faise unha primeira pasada polo texto completo para obter as palabras e a súa frecuencia, e unha segunda pasada na que se substitúe cada palabra polo seu código, xerando así o texto comprimido. Unha solución máis utilizada son os compresores baseados en dicionario, entre os que cabe destacar os da familia Lempel-Ziv [ZL78, ZL77]. As técnicas desta familia procesan o texto por bloques, e obteñen a compresión mediante a substitución de subcadeas o máis longas posibles do texto orixinal, que teñan sido procesadas previamente (habitualmente dentro dunha ventá dun tamaño predefinido), por códigos de lonxitude fixa. Aínda que é posible utilizar estas técnicas para a transmisión de datos (compresión en emisor, transmisión e descompresión en receptor) o feito de traballar por bloques impide a súa utilización en contextos onde a cantidade de datos a transmitir sexa pequena, pero que precise compresión en tempo real. Nestes casos, a alternativa é o uso de técnicas estatísticas dinámicas.

[3] `http://www.7-zip.org/`
[4] `http://www.compression.ru/ds/`

Como o seu nome indica, as técnicas estatísticas dinámicas, tamén chamadas técnicas de unha pasada, realizan un proceso de modelado dinámico, de forma que cada vez que se procesa un símbolo do texto orixinal se actualiza o modelo para saber en todo momento a frecuencia de cada un dos símbolos e o código que lles corresponde. Este proceso realízase tanto no emisor (compresor) como no receptor (descompresor) de forma simétrica. Cada vez que ao emisor lle chega un símbolo, envía o código correspondente a ese símbolo nese intre (tendo en conta a distribución de frecuencias dos símbolos xa procesados e o esquema de codificación usado), e despois actualiza a frecuencia do símbolo e, de ser preciso, o esquema de codificación. O receptor recibe un código, descodifica o símbolo empregando estruturas idénticas ás do emisor, e actualiza tamén a frecuencia do símbolo recibido, mantendo así a simetría (e a sincronización) entre compresor e descompresor.

Unha técnica relevante deste tipo é o Huffman dinámico [Gal78, Vit87]. Neste caso, os símbolos a comprimir son caracteres, e cada vez que aparece o seguinte símbolo é necesario reorganizar a árbore de Huffman binaria para garantir que ningún símbolo máis frecuente ca outro reciba un código máis longo. Este proceso pódese facer en tempo $O(\log \sigma)$, sendo σ altura da árbore de Huffman. En calquera caso, a compresión acadada é pobre (de novo en torno ao 60 % en porcentaxe de compresión). Unha mellora destas técnicas foi o *Dynamic Plain Huffman* e o *Dynamic End-Tagged Dense Code* presentados en [BFNP08]. Estas técnicas que son a versión en "unha pasada" dos seus homólogos semi-estáticos, acadan a mesma porcentaxe de compresión que aqueles, pero son máis lentos en compresión e descompresión debido ao custo extra que supón manter o modelo actualizado cada vez que se procesa unha nova palabra. Neste caso a variante dinámica do *End-Tagged Dense Code* é moito máis rápida pois actualizar o modelo consiste unicamente en levar conta da frecuencia das palabras, e para manter actualizado o esquema de codificación só precisa manter as palabras ordenadas por frecuencia, o que ten un custo de $O(1)$.

En [BFNP05, BFNP10] presentouse unha variante denominada *Dynamic Lightweight End-Tagged Dense Code*. Esta técnica rompe a simetría entre compresor e descompresor de xeito que o descompresor/receptor é moito máis lixeiro pois non ten que levar conta das frecuencias das palabras que recibe. A cambio, o compresor/transmisor debe indicar (cun código especial) cando o código dunha palabra debe cambiar. Na práctica, os autores demostran empiricamente que o número de cambios de código é moi reducido, e polo tanto os códigos asignados ás palabras son moi estables. Isto trae consigo a posibilidade de realizar descompresións moi eficientes, e tamén a posibilidade de realizar buscas dentro do texto comprimido case tan rápidas como as do *End-Tagged Dense Code*. Estas características leváronnos a elixir este compresor dinámico lixeiro para a transmisión de datos na solución para a distribución de contidos dixitais comprimidos en *streaming*, que se explica na sección 7.1.2. Debido a isto, explicaremos en máis detalle o seu funcionamento na sección 3.2.

Se ben o índice invertido é a técnica de indexación máis coñecida e usada, os arrays de sufixos [MM90] son outra das técnicas clásicas no estado da cuestión en indexación. O array de sufixos ($A[1,n]$) mantén unha permutación ordenada lexicograficamente de todos os sufixos $T[i,n]$ do texto indexado $T[1,n]$, de xeito que $T[A[i],n] \preceq T[A[i+1],n]$. Isto permítelle buscar eficientemente calquera patrón $P[1,m]$, non só palabras (indexación de texto completo), en tempo $O(m \log n)$ mediante busca binaria. A cambio as súas necesidades de espazo son altas, habitualmente ao redor de catro veces o tamaño do texto T. Na sección 3.3 explicamos en detalle esta estrutura, que está na base dos autoíndices orientados a palabras que se presentan como contribución da tese no capítulo 4.

Os índices clásicos amosados até o de agora son estruturas auxiliares que permiten facer operacións como contar ou localizar no texto as aparicións dun patrón P, pero que precisan dispor do texto orixinal para poder operar. Unha liña de investigación moi activa na última década enmárcase nos denominados autoíndices [NM07]. Estas son estruturas de datos compactas que, usando espazo próximo á entropía empírica do texto (de orde cero ou superior), conteñen unha representación implícita do texto, que polo tanto xa non se precisa manter á parte, e son capaces de proporcionar buscas eficientes. Exemplos ben coñecidos de autoíndices son o *Compressed Suffix Array de Sadakane* (`CSA`) [Sad03], o *Succint Suffix Array* (SSA) [MN05,FMMN07,MN08,CN08], ou o *FM-Index* [FM00, FM05], entre outros. Ao construír estas estruturas sobre texto en linguaxe natural alcánzanse porcentaxes de compresión que están entre o 50 %–80 % do texto orixinal. Ademais, todos estes autoíndices teñen en común a posibilidade de buscar por calquera subcadea do texto (ao igual que se podía facer nun array de sufixos), o cal os converte en estruturas de busca moi potentes.

Cabe sinalar que no ámbito da linguaxe natural as buscas de texto completo non sempre son imprescindibles, e moitas veces é suficiente poder buscar palabras ou frases. Por ese motivo, e como contribución desta tese (ver capítulo 4), presentamos un autoíndice orientado a palabras adaptado a partir do `CSA` de Sadakane (orientado a caracteres), e que se explicará en detalle na sección 3.4. En concreto deseñamos un Word-Based `CSA` (`WCSA`) [BFN$^+$08, FBN$^+$12] que permite buscar palabras ou secuencias de palabras e obtén porcentaxes de compresión próximas ás que se poden obter con compresores de texto orientados a palabras (entre un 30 % e un 40 %). Ademais, ao buscar directamente secuencias de palabras en lugar de secuencias de caracteres, e dado que o custo da busca vai depender da lonxitude do patrón, as buscas son moito máis eficientes. Tal é así, que mesmo puidemos comprobar empiricamente que estas estruturas son máis rápidas na busca de frases que os índices invertidos orientados a bloques que utilicen un espazo equivalente.

Capítulo 3

Conceptos previos

Neste capítulo explícanse en maior detalle unha serie de técnicas de compresión e indexación que empregamos posteriormente nas estruturas de datos e algoritmos que se presentan como achegas desta tese.

3.1. End-Tagged Dense Code

O *End-tagged Dense Code (ETDC)* [BINP03, BFNP07] é un compresor baseado en palabras e orientado a bytes que xurdiu en 2003 tratando de mellorar a compresión acadada polo *Tagged Huffman Code (TH)* [MNZB98, MNZBY00], e mantendo as propiedades que o convertían na técnica de referencia entre o estado da cuestión da compresión para Bases de Datos textuais. Recordemos que TH é unha técnica de compresión orientada a bytes que reserva un bit de cada byte para indicar se o byte era o primeiro ou non dentro dun código. Ademais, para garantir que os códigos sexan libres de prefixo, utiliza codificación Huffman cos restantes 7 bits de cada byte.

O ETDC xorde cun pequeno cambio sobre o TH: en lugar de usar un bit de marca para indicar o inicio dun código, márcase o último byte de cada código. Esta aparentemente intranscendente modificación trae consigo a propiedade de que os códigos se converten automaticamente en códigos libres de prefixo, independentemente do contido dos 7 bits restantes de cada byte. Polo tanto, non é necesario utilizar codificación Huffman para garantir esta propiedade, e pódense utilizar todas as 2^7 combinacións posibles obtendo así unha codificación densa. De feito, se partimos dun vocabulario de palabras xa ordenado descendentemente por frecuencias, e se temos en conta que o primeiro bit do último byte de cada código se

estableza a 1 e o primeiro bit dos restantes bytes se poña a 0, o resto do proceso de codificación sobre os restantes bits de cada byte é totalmente secuencial, e non depende da frecuencia das palabras, senón unicamente da súa posición dentro do vocabulario. Desta forma a primeira palabra do vocabulario ($i = 0$) é codificada como $\underline{1}0000000$, a segunda recibe o código $\underline{1}0000001$, e así secuencialmente até a palabra na posición 127 que recibe o código $\underline{1}1111111$. Rematados os códigos de 1 único byte, a palabra 129ª ($i = 128$) será codificada como $\underline{0}0000000{:}\underline{1}0000000$, a seguinte como $\underline{0}0000000{:}\underline{1}0000001$ e así sucesivamente até a palabra que está na posición $(128^2+128-1)$ que recibirá o código $\underline{0}1111111{:}\underline{1}1111111$. A seguinte palabra (posición $128^2 + 128$ no vocabulario) recibirá o primeiro dos códigos de tres bytes $\underline{0}0000000{:}\underline{0}0000000{:}\underline{1}0000000$, e así sucesivamente. É doado ver que o número de palabras codificadas con 1, 2, 3 etc., bytes é fixo, e concretamente é de 128, 128^2, 128^3 etc.

No cadro 3.1, inclúese un exemplo de asignación de códigos ás palabras dun vocabulario asumindo que usamos TH e ETDC sobre bytes "especiais" de só 3 bits. Pódese ver que nese caso o número de códigos de 1 byte en ETDC é 4 e o número de códigos de 2 bytes sería 16 (aínda que non se utilizan todos). Por outra banda, pódese ver que TH desaproveita o código de 1 byte [111] para poder crear códigos máis longos, e o mesmo sucede co código de dous bytes [111][011]. Á dereita tamén se pode apreciar como a lonxitude media dos códigos xerados por ETDC é menor.

Cadro 3.1: Comparación da codificación Tagged Huffman e a End-Tagged Dense Code. Asúmese que os nosos bytes "só teñen 3 bits" e que a distribución de frecuencias dos símbolos é a indicada na terceira columna.

					Frec × bytes	
Pos	Palabra	Frec	ETDC	TH	ETDC	TH
0	A	0,200	[100]	[100]	0,200	0,200
1	B	0,200	[101]	[101]	0,200	0,200
2	C	0,150	[110]	[110]	0,150	0,150
3	D	0,150	[111]	[111][000]	0,150	0,300
4	E	0,140	[000][100]	[111][001]	0,280	0,280
5	F	0,090	[000][101]	[111][010]	0,180	0,180
6	G	0,040	[000][110]	[111][011][000]	0,080	0,120
7	H	0,020	[000][111]	[111][011][001]	0,040	0,060
8	I	0,005	[001][100]	[111][011][010]	0,010	0,015
9	J	0,005	[001][101]	[111][011][011]	0,010	0,015
			Lonxitude Media do Código		**1,300**	**1,520**

O feito de que o código asignado a unha palabra dependa unicamente da súa

posición dentro do vocabulario, non da súa frecuencia, fai que sexa posible crear algoritmos de codificación e descodificación moi simples e rápidos. Na práctica, sendo i a posición dunha palabra p_i no vocabulario, e C_i o código que lle corresponde, existe un algoritmo de codificación (similar a un cambio de base) $C_i \leftarrow encode(i)$ que en tempo $O(|C_i|)$ obtén o código correspondente á palabra p_i. Ademais tamén dispoñemos do algoritmo de descodificación $i \leftarrow decode(C_i)$ que, dado un código, o descodifica obtendo a posición no vocabulario correspondente á palabra p_i codificada co código C_i tamén en tempo $O(|C_i|)$.

Ademais, e xa que a frecuencia pouco importa unha vez que temos as palabras ordenadas por frecuencia, ao comprimir un texto só debemos incluír xunto ao texto comprimido o listado de palabras do vocabulario ordenadas por frecuencia. Con esa información, e aplicando $i \leftarrow decode(C_i)$, o descompresor ETDC será capaz de recuperar o texto orixinal.

Por outra banda, o feito de usar un bit de marca confírelle ao ETDC as mesmas capacidades de busca directa sobre texto comprimido das que dispoñía o TH. Isto é, capacidade para realizar *string matching* utilizando os algoritmos da familia Boyer-Moore, que son os máis rápidos na práctica. Igualmente, ese bit de marca tamén garante que os códigos ETDC se autosincronicen, e polo tanto sexa posible realizar acceso directo e descompresión aleatoria desde calquera posición do texto comprimido.

Os experimentos realizados co ETDC [BFNP07] sitúano entre os mellores compresores para o ámbito das bases de datos textuais, avantaxando ao TH en aproximadamente 3 puntos porcentuais, e mantendo todas as súas boas cualidades. Ademais os seus simples algoritmos de codificación e descodificación convérteno nunha técnica rápida e flexible. Como veremos na seguinte sección é moi doado obter un compresor dinámico baseado no ETDC.

3.2. Compresores dinámicos baseados no ETDC

Tendo en conta as facilidades dos algoritmos directos de codificación e descodificación do ETDC, é relativamente sinxelo crear unha versión dinámica ou nunha soa pasada do ETDC. Esta técnica denominouse *Dynamic End-Tagged Dense Code (DETDC)* e traballa como os compresores dinámicos tradicionais [Gal78, Vit87]. Posteriormente, unha mellora do DETDC denominada *Dynamic Lightweight End-Tagged Dense Code (DLETDC)* rompe a simetría típica entre compresor e descompresor nas técnicas dinámicas para acadar un descompresor máis lixeiro e rápido. Describimos brevemente o funcionamento destas técnicas de seguido.

3.2.1. DETDC

O *Dynamic End-Tagged Dense Code (DETDC)* [BFNP04, BFNP08] funciona como un compresor dinámico tradicional. Tanto o emisor como o receptor manteñen o modelo sincronizado (neste caso, simplemente o vocabulario coas palabras ordenadas por frecuencia). Comezando cun vocabulario baleiro, durante a compresión, cando o emisor procesa unha palabra que está situada na posición i do vocabulario ordenado, utiliza $C_i \leftarrow encode(i)$ para obter o código que lle corresponde a dita palabra e envía ese código. Cando o receptor o recibe, aplica $i \leftarrow decode(C_i)$, e obtén así a posición da palabra que foi transmitida no vocabulario ordenado. A continuación, tanto emisor como receptor realizan o mesmo proceso de actualización para manter o vocabulario ordenado tras ter procesada a palabra actual. Na práctica este proceso consiste en sumarlle 1 á frecuencia f da palabra recibida, e posto que cambia de frecuencia f a $f+1$, hai que colocala como última palabra de frecuencia $f+1$. Para facer isto unicamente hai que facer un *swap* entre a palabra procesada e a primeira palabra con frecuencia f. O custo total é $O(1)$. Para máis detalles véxase [BFNP08].

Por outra banda cando o emisor le unha palabra que non ten aparecido previamente no texto (e polo tanto aínda non está no vocabulario), infórmalle ao receptor desta situación enviando un código especial que indica "vén unha nova palabra" (C_{new}) seguido da palabra en plano. A continuación engádese esa palabra ao final do vocabulario.

3.2.2. DLETDC

O DETDC é unha técnica simple, máis eficiente que unha técnica baseada en Huffman homóloga [BFNP08], pero a súa rapidez en canto a descompresión e busca pode ser mellorada. O *Dynamic Lightweight End-Tagged Dense Code (DLETDC)* [BFNP05, BFNP10] xurdiu con ese obxectivo.

O DLETDC basease no DETDC, pero evita a sobrecarga que conleva manter o modelo actualizado na parte do receptor, que fai que por cada palabra recibida haxa que facer unha reorganización do vocabulario. Isto é moi conveniente cando temos un escenario cun ancho de banda limitado, ou temos un receptor lixeiro con unha capacidade de procesamento e memoria limitadas.

No caso do DLETDC o emisor é o encargado de levar conta da frecuencia dos símbolos que leven chegado até un determinado instante, e de manter o vocabulario ordenado por frecuencia. Pola súa banda o receptor simplemente almacenará un vocabulario de palabras (non ten que contabilizar frecuencias). Cando unha palabra p_i chega, o receptor simplemente a ten que descodificar utilizando $i \leftarrow decode(C_i)$, e así obtén a posición na que está actualmente no vocabulario. O receptor non ten que facer nada máis: nin contabilizar frecuencias, nin reordenar as palabras

do vocabulario. Así que o emisor é o encargado de avisar ao receptor se nalgún momento a correspondencia dun código a unha palabra debe cambiar. Nótese que os cambios na asignación de códigos tras determinados cambios de frecuencia poden ser necesarios para que se manteñan porcentaxes de compresión óptimas.

A chave para que isto sexa viable é que o número de cambios a notificar non sexa moi elevado. Este feito queda patente se nos fixamos en que non é necesario cambiar o código que lle corresponde a unha palabra cada vez que esta palabra chega, incrementando a súa frecuencia en +1, xa que ese cambio de código non melloraría a compresión debido a que na maioría dos casos non cambia o número de bytes do código que se lle asigna á palabra. Deste modo a idea é que só se modificará a correspondencia palabra $\leftrightarrow$ código cando o incremento na frecuencia da palabra p_i faga preciso codificala cun código máis curto que o código C_i que teña actualmente asignado. Nótese que agora ese código non depende da posición que a palabra teña no vocabulario, senón que vai ser exactamente o mesmo código que se lle asignou por vez primeira (salvo que xa se teña modificado). Así, sempre que chegue a mesma palabra p_i, envíase o mesmo código C_i, salvo cando esa palabra se fai merecedora de utilizar un código máis curto. Nese caso, faise un *swap* desa palabra con aquela (p_x) que teña a menor frecuencia de todas as que se codificaban cun byte menos. Como resultado do *swap* os códigos asignados a esas palabras p_i e p_x son intercambiados, pero ningunha outra palabra do vocabulario se ve implicada en dito cambio. O emisor neste caso envía un código especial C_{swap} seguido dos códigos das palabras que se intercambian C_i C_x. Ou o que é o mesmo, $\langle C_{swap}, C_i, C_x \rangle$ indica que a partir de agora a palabra p_i será codificada con C_x, e a palabra p_x con C_i. Neste caso o receptor simplemente ten que intercambiar as palabras que están nas posicións i, x do seu vocabulario.

Ao igual que no DETDC cando chega unha palabra nova envíase C_{new} seguida da palabra en plano, e o receptor inserta dita palabra na última posición do vocabulario.

Nótese que DLETDC precisa dos códigos especiais C_{new} e C_{swap}. En [BFNP10] os autores consideraron que a mellor opción (descodificación e buscas máis rápidas e perda imperceptible de compresión) é utilizar respectivamente os dous últimos códigos de 3 bytes dispoñibles na codificación ETDC. Isto é, os códigos 0111111:0111111:11111110 e 0111111:0111111:11111111.

Finalmente, en [BFNP10] presentáronse evidencias experimentais e analíticas de que o número de palabras que cambian de código durante o proceso de compresión é moi baixo. Ademais apréciase que a maioría deses cambios (90 %) se producen mentres temos procesado non máis do 10 % da colección. Isto fai que o texto comprimido/enviado con DLETDC poida ser buscado facilmente. Nunha primeira fase da busca, e usando un algoritmo *Set-Horspool multipatrón* [NR02], buscaranse os patróns C_{new} e a palabra que desexemos atopar en plano. No momento en que atopemos dita palabra, o número de veces que teña aparecido C_{new} indicaranos o

código C_i que lle corresponde á nosa palabra. A partir dese momento, buscaremos en paralelo o código C_i e C_{swap}. O código C_i pode ter que cambiar, pero, nese caso, ese cambio irá avisado cun C_{swap} que tamén estamos buscando.

3.3. Array de sufixos

Dentro dos índices de texto completo é preciso destacar, pola súa importancia, o array de sufixos de Manber e Myers [MM93]. O array de sufixos é un índice básico que permite buscar rapidamente un patrón nun texto mediante busca binaria. Toda busca binaria está baseada na existencia dunha relación de orde entre os elementos a buscar, que neste caso son cadeas de caracteres. A relación de orde considerada é a orde lexicográfica entre cadeas, que se define formalmente como segue: Sexan a e b caracteres e X e Y cadeas. Entón $aX < bY$ se $a < b$, ou se $a = b$ e $X < Y$. Por outra parte, $\varepsilon < X$ para calquera $X \neq \varepsilon$ (onde ε representa a cadea baleira). Sexa T un texto de lonxitude n. Resulta sinxelo ver que hai n sufixos distintos de T, onde cada un dos n caracteres de T é o primeiro carácter dun sufixo. O array de sufixos A do texto T é a secuencia formada polos seus n sufixos ordenados lexicograficamente. O que realmente se almacena no array de sufixos non é o sufixo en si, senón unha referencia a el. Un sufixo do texto queda perfectamente definido mediante un índice que represente a posición no texto na que dito sufixo comeza. Este índice será o que se almacene no array de sufixos.

Para ilustrar as estruturas desta e das seguintes seccións utilizaremos o seguinte texto de exemplo: $T =$ "`cava o cabo na cova$`", onde $n = 20$. Asumimos que o texto remata cun carácter especial '$', que é menor que todos os demais caracteres do alfabeto, e que non aparece en ningún outro lugar do texto nin do patrón. Ademais, por claridade, imos representar o espazo en branco cun guión baixo '_', que será considerado menor lexicograficamente a calquera carácter alfanumérico.

Na figura 3.1 móstrase o array de sufixos para o texto do exemplo. Debaixo de cada un dos elementos do array de sufixos aparece representado o sufixo do texto ao cal apunta. Como se pode ver, estes sufixos están ordenados lexicograficamente, e polo tanto os sufixos que comezan polo mesmo carácter (e máis xeralmente polo mesmo prefixo) aparecen consecutivos.

O problema da busca dun patrón nun texto pode verse como a tarefa de atopar os sufixos do texto que teñen como prefixo a dito patrón, é dicir, que comecen por el. O feito de que os sufixos estean ordenados permite o emprego da busca binaria, que é unha técnica de busca moi eficiente.

A continuación descríbese o procedemento de busca dun patrón nun texto

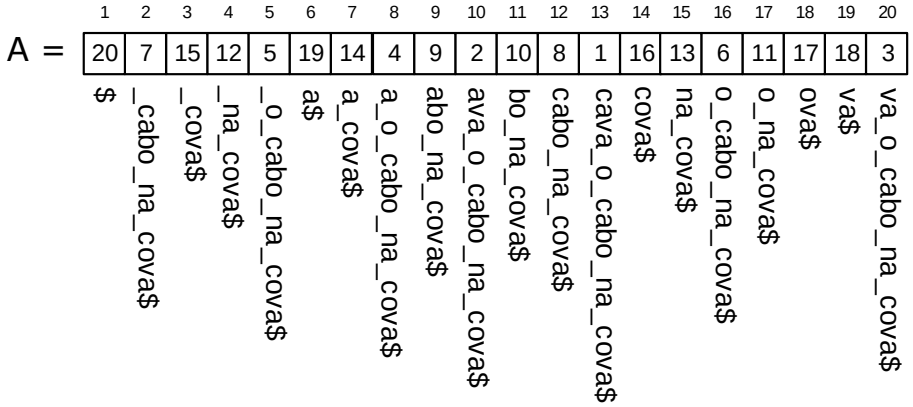

Figura 3.1: Array de sufixos sobre o texto "`cava o cabo na cova$`".

mediante o emprego do array de sufixos. Selecciónase inicialmente o elemento central do array, accédese á posición do texto á que apunta e compárase lexicograficamente o patrón buscado co texto a partir desa posición. Se o patrón é maior que dito texto, repítese o proceso na segunda metade do array de sufixos, e se o patrón é menor, continúase a busca na primeira metade. Salientar que, se o patrón aparece no texto, os sufixos que comezan por dito patrón van aparecer en posicións consecutivas do array de sufixos. Realizando dúas buscas binarias (unha para atopar o límite inferior e outra para o límite superior) é posible determinar o rango do array de sufixos que contén os sufixos que comezan polo patrón buscado. Como se pode ver, o array de sufixos acelera as buscas sobre o texto pero segue precisando botar man del para operar.

Para rematar coa exposición do array de sufixos, proporciónase unha definición formal do mesmo.

Definición 1: *O array de sufixos dun texto* $T_{1,n}$ *é un array* $A[1..n]$ *que contén unha permutación do intervalo* $[1, n]$*, de forma que* $T_{A[i],n} < T_{A[i+1],n}$ *para todo* $1 \leq i < n$*, onde a relación de orde entre cadeas é a orde lexicográfica.*

O array de sufixos proporciona unha gran eficiencia nas buscas, cimentada na velocidade da busca binaria, pero presenta a limitación de precisar unha gran cantidade de espazo de almacenamento para operar. O excelente rendemento que o array de sufixos consegue nas buscas dou lugar a que se investigase (e se siga

investigando) no desenvolvemento de distintos índices baseados nel. A idea central é a de manter o método de busca do array de sufixos, pero reducindo a cantidade de espazo precisado mediante distintas técnicas de compresión.

Da idea de comprimir o array de sufixos xorden distintos índices (xeralmente coñecidos como arrays de sufixos comprimidos) entre os que imos destacar dous: o array de sufixos compacto de Mäkinen [Mäk00] e o array de sufixos comprimido de Grossi e Vitter [GV00]. Entraremos en maior profundidade na explicación deste último xa que, como se verá, introduce unha estrutura (a función Ψ) que será básica para o autoíndice construído posteriormente por Sadakane, que á súa vez é a base do noso autoíndice orientado a palabras, que se presenta no capítulo 4. Os arrays de sufixos comprimidos son índices que manteñen o algoritmo de busca dos arrays de sufixos, pero substitúen o array por unha estrutura de datos que ocupa menos espazo e que permite acceder aos elementos do array de sufixos orixinal. Os índices de Mäkinen e o de Grossi e Vitter apareceron simultaneamente e de xeito independente durante o ano 2000.

3.3.1. Array de sufixos compacto de Mäkinen

Os arrays de sufixos non son permutacións aleatorias. Cando o tamaño do alfabeto co que se constrúe un texto (σ) é menor que o tamaño do texto (n), non todas as permutacións de $[1, n]$ poden ser o array de sufixos dalgún texto (debido a que existen máis permutacións que textos de tamaño n). Esta idea de non aleatoriedade do array de sufixos suxire a existencia dalgunha característica do mesmo (regularidade) que permita a súa compresión para representalo empregando menos espazo.

Un texto posúe unha propiedade intrínseca, coñecida como compresibilidade, que define a súa capacidade de compresión. A compresibilidade dun texto vén determinada pola situación concreta dos símbolos que o constitúen, ou o que é o mesmo, polo propio texto. Pensemos en dous textos do mesmo tamaño, construídos sobre o mesmo alfabeto e nos cales aparecen os mesmos caracteres e coa mesma frecuencia, aínda que en distinta disposición: $\Sigma = [a, b, c, d]$, $T_1 =$"*aaabbbbbcccddd*", e $T_2 =$ "*bdabcacdabdbcb*". Se agora tentamos memorizar ambos textos, decatámonos que nos resulta máis doado memorizar T_1 que T_2. A simple vista, e aínda sen entender moi ben a razón, semella que o texto T_1 pode ser representado (con algún tipo de técnica de compresión) empregando menos información que o texto T_2. A razón é que, efectivamente, o texto T_1 contén menos información que o texto T_2 e polo tanto é máis compresible. Unha medida da información que contén un texto é a entropía empírica de orde k, que non é máis que a aplicación do concepto clásico de entropía de orde k a textos finitos. Sen ánimo de profundar máis neste concepto, destacar que existe unha relación entre a entropía empírica de orde k dun texto e as regularidades que aparecen no seu array de sufixos. Basicamente, o que isto significa é que existe unha relación directa entre a compresibilidade dun texto e a do seu array de sufixos.

Mäkinen [Mäk00] define un concepto, o de autorrepetición, que logo empregará para medir a regularidade existente nun array de sufixos. Unha autorrepetición defínese como un intervalo do array de sufixos que se repite, desprazado unha unidade, noutro lugar do mesmo. As autorrepeticións xorden nos arrays de sufixos pola existencia de cadeas de caracteres repetidas no texto orixinal. Non obstante, non todas as cadeas de caracteres que se repiten no texto van dar orixe a autorrepeticións. Estas só aparecen cando para un par de sufixos aX e aY consecutivos en A, os sufixos X e Y son tamén consecutivos en A, ou o que é o mesmo, non existe ningún sufixo Z no texto tal que $X < Z < Y$. A definición formal deste concepto é a seguinte:

Definición 2: *Dado un array de sufixos A, unha autorrepetición é un intervalo $[i, i+l]$ de $[1, n]$ tal que existe outro intervalo $[j, j+l]$ tal que $A[j+r] = A[i+r]+1$ para todo $0 \leq r \leq l$. Por conveniencia técnica, consideramos que a cela $A[1] = n$ é unha autorrepetición dela mesma, de lonxitude 1.*

Mäkinen establece como medida da regularidade existente nun array de sufixos o número mínimo de autorrepeticións (intervalos non solapados) precisas para cubrir o array (representamos este número por n_{sr}). Canto menor sexa n_{sr} maior será a regularidade do array e maior tamén a súa compresibilidade.

O array de sufixos compacto de Makinen [Mäk00] foi especificamente deseñado para explotar as autorrepeticións existentes no array de sufixos. A súa idea é a de dividir dito array no número mínimo de autorrepeticións precisas para cubrilo, ou o que é o mesmo, en n_{sr} intervalos non solapados. Cada un destes intervalos será representado por unha estrutura de datos chamada bloque. Deste xeito, o array de sufixos orixinal é transformado nunha secuencia de bloques que contén a suficiente información para reconstruílo.

Non entraremos en detalle na descrición da estrutura do array de sufixos compacto de Mäkinen, xa que o noso autoíndice non empregará esta idea. Sen embargo, consideramos interesante a súa introdución para presentar a relación existente entre a compresibilidade dun texto (determinada pola súa entropía empírica de orde k) e a do seu array de sufixos (determinada polo número mínimo de autorrepeticións precisas para cubrilo).

Máis interesante para nós é o array de sufixos comprimido de Grossi e Vitter [GV00], xa que emprega unha función (a función Psi) para descompoñer xerarquicamente o array de sufixos co fin de reducir o seu espazo de almacenamento. Máis adiante veremos como esta función vai permitir prescindir do texto orixinal, sendo básica para o noso autoíndice.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A =	20	7	15	12	5	19	14	4	9	2	10	8	1	16	13	6	11	17	18	3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Ψ =	13	12	14	15	16	1	3	5	11	20	17	9	10	18	7	2	4	19	6	8

Figura 3.2: Función Ψ para o texto "`cava o cabo na cova$`".

3.3.2. Array de sufixos comprimido de Grossi e Vitter

O array de sufixos comprimido de Grossi e Vitter [GV00] emprega unha descomposición xerárquica do array de sufixos. Para efectuar dita descomposición, Grossi e Vitter definen unha importante función á que denominan Ψ. O que a función Ψ devolve é, para cada posición do array de sufixos, cal é a posición no mesmo que apunta ao seguinte sufixo do texto orixinal. A continuación incluímos a definición formal e un exemplo na figura 3.2 que axuda a entender o concepto.

Definición 3: *Dado un array de sufixos $A[1,n]$, a función $\Psi : [1,n] \rightarrow [1,n]$ defínese de forma que, para todo $1 \leq i \leq n$, $A[\Psi(i)] = A[i] + 1$. A única excepción é $A[1] = n$, xa que nese caso precisamos que $A[\Psi(1)] = 1$, co que Ψ será realmente unha permutación.*

A continuación explícase polo miúdo a estrutura do índice de Grossi e Vitter. Centrámonos no primeiro nivel da descomposición xerárquica. Sexa $A_0 = A$ o array de sufixos orixinal, de tamaño n. Definimos un vector de bits $B_0[1,n]$ tal que $B_0[i] = 1$ se, e só se, $A[i]$ é par. Sexa ademais $\Psi_0[1, \lceil n/2 \rceil]$ a secuencia de valores $\Psi(i)$ para os valores de i onde $B_0[i] = 0$. Finalmente, sexa $A_1[1, \lfloor n/2 \rfloor]$ a subsecuencia de $A_0[1,n]$ formada polos valores pares $A_0[i]$ divididos por 2. Entón, $A = A_0$ pode ser representado empregando só Ψ_0, B_0 e A_1.

Antes de explicar o método para obter $A[i]$ a partir de Ψ_0, B_0 e A_1, imos introducir unha importante función, que é a función *rank* aplicada a secuencias binarias. Dado un vector de bits B e unha posición i dentro del, a función *rank* devolve o número de bits dun determinado valor que hai en B até a posición i. Existen polo tanto dúas versións da función *rank* aplicada a secuencias binarias: a que conta o número de 0s ($rank_0$) e a que conta o número de 1s ($rank_1$).

Para recuperar $A[i]$ a partir de Ψ_0, B_0 e A_1, temos que mirar primeiro se $B_0[i] = 1$. Se é así, $A[i]$ aparece (dividido por 2) nalgún lugar de A_1. A posición exacta depende de cantos 1s hai en B_0 até a posición i, é dicir, $A[i] = 2 \cdot A_1[rank_1(B_0, i)]$. Se

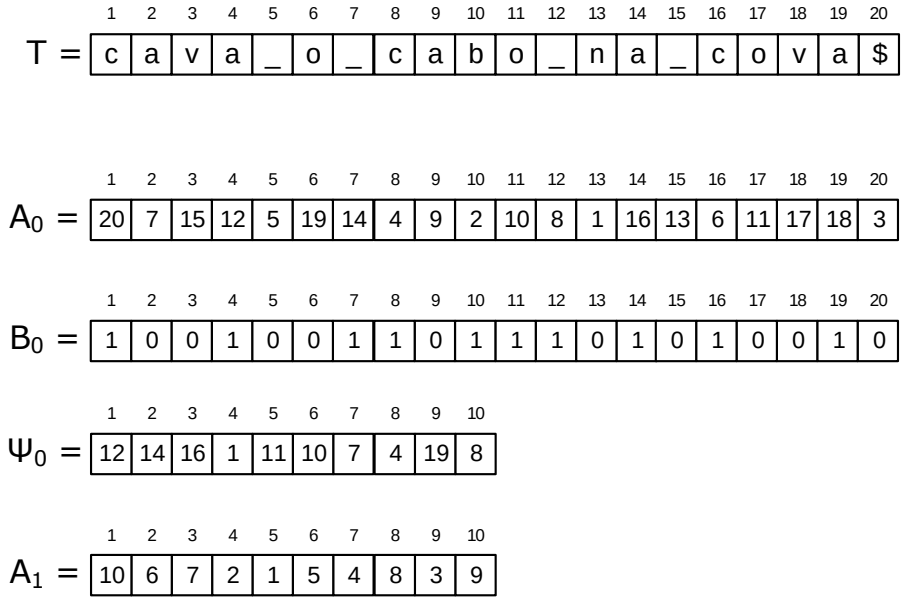

Figura 3.3: Primeiro nivel da estrutura recursiva de Grossi e Vitter para o texto "`cava o cabo na cova$`".

$B_0[i] = 0$, entón $A[i]$ é impar e non está en A_1. Non obstante, $A[i]+1 = A[\Psi(i)]$ ten que ser par e polo tanto aparecer en A_1. Como Ψ_0 contén só os valores de Ψ onde $B_0[i] = 0$, temos que $A[\Psi(i)] = A[\Psi_0[rank_0(B_0, i)]]$. Unha vez calculado $A[\Psi(i)]$ (para o $\Psi(i)$ par), obtemos $A[i]$ simplemente como $A[i] = A[\Psi(i)] - 1$.

Esta idea pode ser empregada recursivamente: en lugar de representar directamente A_1, podemos descompoñelo e substituílo por Ψ_2, B_2 e A_2. Podemos continuar até que A_h sexa o suficientemente pequeno como para ser almacenado explicitamente.

Na figura 3.3 móstrase un exemplo do primeiro nivel da estrutura recursiva de Grossi e Vitter. Vexamos agora como podemos obter os elementos do array de sufixos orixinal a partir da súa descomposición en Ψ_0, B_0 e A_1. Mostraremos dous exemplos, un para obter un valor par e outro para obter un valor impar.

Supoñamos que queremos obter $A[8]$. O primeiro que temos que facer é acceder á posición 8 do vector de bits B e ver o valor do bit que almacena. Neste caso é un 1 ($B[8] = 1$) o que nos indica que o valor de $A[8]$ é par e que polo tanto aparece, dividido por 2, no array A_1. Para saber en que posición de A_1 aparece este valor, precisamos saber cantos valores pares hai no array de sufixos até a posición 8, o

que coincide co número de 1s que hai en B_0 até dita posición. Para realizar esta conta, empregamos a función $rank_1$. Como $rank_1(B_0, 8) = 4$ sabemos que o valor almacenado na posición 8 do array de sufixos ocupa a cuarta posición relativa dos valores pares do mesmo. Para rematar só falta acceder á posición cuarta de A_1 e multiplicar o seu valor por 2, obtendo que $A[8] = 2 \cdot A_1[4] = 2 \cdot 2 = 4$.

Vexamos agora o procedemento para obter o valor de $A[6]$. Procedemos do mesmo xeito que antes, pero neste caso observamos que $B[6] = 0$, o que nos indica que o valor que buscamos é impar e polo tanto non pode ser obtido directamente de A_1. Non obstante, $A[6] + 1$ si que ten que ser necesariamente par. Pola definición da función Ψ, sabemos que $A[\Psi(i)] = A[i] + 1$, logo $A[\Psi(6)] = A[6] + 1$. Se nos lembramos, o vector Ψ_0 almacena o valor da función Ψ para aquelas posicións do array de sufixos que conteñen valores impares, e polo tanto ten que almacenar o valor de $\Psi(6)$. Para saber en que posición de Ψ_0 se atopa o valor $\Psi(6)$ precisamos desta vez saber cantos valores impares hai no array de sufixos até a posición 6, o que coincide co número de 0s que hai até dita posición en B_0. Para contalos empregamos agora a función $rank_0$ que nos indica que son 4 ($rank_0(B_0, 6) = 4$). Accedemos agora ao cuarto elemento de Ψ_0 e vemos que $\Psi_0[4] = 1 = \Psi(6)$. Recapitulando temos que $A[\Psi(6)] = A[1] = A[6] + 1$. Agora tan só falta calcular $A[1]$ (observamos que ten que ser necesariamente un valor par) como se explicou no parágrafo anterior e despexar o valor de $A[6]$. Evitamos aquí repetir todo o proceso para o cálculo de $A[1] = 20$. Finalmente obtemos que $A[6] = A[1] - 1 = 20 - 1 = 19$.

Chegados a este punto, resulta oportuno lembrar que, tanto o array de sufixos compacto de Mäkinen como o array de sufixos comprimido de Grossi e Vitter, manteñen o procedemento de busca orixinal do array de sufixos, o que significa que seguen precisando dispor do texto orixinal para funcionar. Sadakane foi máis aló e demostrou como é posible empregar a función Ψ para construír un índice que permita prescindir do texto orixinal.

3.4. Autoíndice de Sadakane

A diferenza dos arrays de sufixos comprimidos vistos anteriormente, o autoíndice de Sadakane [Sad03] non proporciona acceso directo ao array de sufixos, sen embargo, si que permite acceder a calquera prefixo de $T_{A[i],n}$. É dicir, dada unha entrada do array de sufixos, non podemos saber en que posición do texto se atopa o sufixo apuntado por ela, pero si podemos saber que sufixo é. Desta forma, podemos efectuar a comparación entre o patrón buscado e o sufixo apuntado por unha entrada do array de sufixos, aínda que non coñezamos a posición de dito sufixo no texto (ao fin e ao cabo, tampouco imos dispor do texto para acceder a ela). Isto vainos permitir conservar o mesmo procedemento de busca que empregaba o array de sufixos, só

que agora non imos acceder explicitamente ao texto para obter o sufixo a comparar co patrón. De seguido introdúcense as estruturas que compoñen o autoíndice de Sadakane.

Sexa A o array de sufixos e T o texto orixinal, o autoíndice de Sadakane vai representar A e T empregando a función Ψ (definida por Grossi e Vitter) e unha estrutura adicional. Esta estrutura adicional estará formada por un vector de bits D e pola secuencia ordenada S dos caracteres que aparecen no texto. Con ela será posible coñecer, dada unha entrada do array de sufixos, cal é o primeiro carácter do sufixo ao que dita entrada apunta. D é un vector de bits que marca as posicións do array de sufixos nas cales se produce un cambio no primeiro carácter do sufixo apuntado. Pola definición de array de sufixos explicada anteriormente, os sufixos que comezan polo mesmo carácter van ser apuntados por posicións consecutivas do array de sufixos. A primeira destas posicións será marcada cun 1 en D, e todas as demais cun 0.

Na figura 3.4 mostramos un exemplo que permitirá entender mellor estes conceptos. Aínda que na figura aparecen as estruturas T e A correspondentes ao texto e ao array de sufixos respectivamente, as estruturas do autoíndice de Sadakane son unicamente Ψ, D e S.

Unha vez vistas as estruturas do autoíndice, imos ver como empregalas para obter o sufixo apuntado por unha determinada posición do array. Supoñamos que queremos obter o sufixo apuntado pola posición 12 do array de sufixos do exemplo, e dicir $T_{A[12],n}$.

Comezamos calculando o primeiro carácter do sufixo. Para elo empregamos a estrutura formada por D e S. O número de 1s que hai no vector D até a posición i indica cantos caracteres distintos son cabeza dos sufixos apuntados polas posicións $1 \dots i$ do array. Esta conta realízase coa función $rank_1$. Dado que os sufixos se atopan ordenados lexicograficamente, resulta evidente que os seus primeiros símbolos tamén o están. Deste xeito, podemos empregar o resultado anterior de $rank_1$ para acceder a S e obter o símbolo inicial do sufixo buscado, $T_{A[i]} = S[rank_1(D, i)]$. Así, o primeiro carácter do sufixo $T_{A[12],n}$ sería $T_{A[12]} = S[rank_1(D, 12)]$, e como $rank_1(D, 12) = 5$ e $S[5] =$'c', concluímos que o sufixo apuntado pola posición 12 do array comeza polo carácter 'c'.

Agora precisamos obter o seguinte símbolo do sufixo, que sería $T_{A[12]+1}$. Neste punto é onde botamos man da función Ψ, que nos vai permitir avanzar unha posición no texto. Volvendo a definición de Ψ observamos que $A[\Psi(12)] = A[12] + 1$, e polo tanto $T_{A[12]+1} = T_{A[\Psi(12)]}$. Como $\Psi(12) = 9$, temos que $T_{A[12]+1} = T_{A[9]}$. Calculamos agora $T_{A[9]}$ como se explicou anteriormente ($T_{A[9]} = S[rank_1(D, 9)]$) e obtemos que o segundo carácter do sufixo é 'a'.

Podemos seguir extraendo caracteres do sufixo até chegar o final do texto. É importante destacar que non todas as comparacións entre un patrón e un sufixo van precisar extraer o sufixo completo. Só será necesario extraer caracteres do sufixo até poder establecer a relación lexicográfica entre o patrón e dito sufixo. Así, se o primeiro carácter dun sufixo non coincide co primeiro carácter do patrón, xa non habería que seguir obtendo caracteres do sufixo, pois xa poderíamos determinar se o patrón é maior ou menor que el (obviamente será distinto).

Co método descrito para a extracción de caracteres dun sufixo, resulta posible continuar obtendo caracteres máis aló do final do texto. Isto é debido a que a función Ψ enlaza o derradeiro carácter do texto co primeiro (véxase a excepción feita na definición 3). Neste punto xorde un pequeno problema. Supoñamos que o primeiro carácter do texto é 'A' e o derradeiro é 'a'. Como a función Ψ enlaza de forma cíclica o final do texto co seu principio, o método de busca descrito considera que a 'a' coa que remata o texto vai seguida da 'A' coa que dito texto comeza. Deste xeito, a busca do patrón $P = aA$ atoparía aquí unha coincidencia, cando realmente dito patrón non aparece no texto orixinal (ou polo menos, non nesa posición). É por isto polo que se emprega o carácter especial '$' como derradeiro carácter dun texto. O carácter '$' é inferior lexicograficamente a todos os demais caracteres do alfabeto e, como non pode aparecer no patrón, provoca que se deteña a comparación ao chegar a el evitando que se sigan extraendo caracteres polo principio.

Unha vez visto o exemplo e entendido o seu funcionamento, móstranse unha serie de limitacións que presenta o autoíndice de Sadakane así definido. En primeiro lugar está o feito xa comentado de que este índice non proporciona acceso directo a $A[i]$, de forma que non resulta posible resolver a tarefa de localizar (*locate*) as aparicións do patrón no texto. O que o índice construído permite coñecer é o intervalo de posicións do array de sufixos que apuntan a sufixos que comezan polo patrón buscado. Así, resólvese a tarefa de contar (*count*) o número de aparicións do patrón no texto, que non será máis que a lonxitude de dito intervalo. Mais como non dispoñemos do array de sufixos, non temos suficiente información para saber as posicións no texto ás que apuntan as entradas contidas no intervalo. Outra limitación non menos importante é que coas estruturas definidas non é posible visualizar (*display*) unha subcadea do texto dadas a súa posición inicial e final. Esta tarefa resulta imprescindible para un autoíndice, xa que debe permitir prescindir do texto orixinal.

Para poder superar estas limitacións, resulta necesario que o autoíndice proporcione acceso ao array de sufixos (para a tarefa de *locate*) e tamén á súa inversa (para a tarefa de *display*). A inversa do array de sufixos, A^{-1} permite saber cal é a posición no array de sufixos que ocupa un sufixo a partir da posición que dito sufixo ocupa no texto orixinal. Unha vez obtida a posición no array de sufixos, pode visualizarse o fragmento do texto apuntado, obtendo os caracteres polo método xa descrito.

E aquí onde Sadakane fai uso da estrutura xerárquica de Grossi e Vitter, que proporcionaba acceso ao array de sufixos, pero sen o texto e coa función completa Ψ en lugar de Ψ_0 (dado que xa precisaba ter Ψ completa para extraer caracteres do texto). Non obstante, a estrutura de Grossi e Vitter non vai ser directamente aplicable ao noso autoíndice orientado a palabras. Basicamente, a razón disto radica en que no caso do índice de texto completo (o visto até agora), os caracteres teñen lonxitude constante, mentres que no caso do índice de palabras, estas van a ter unha lonxitude variable. Como consecuencia disto, o autoíndice orientado a palabras vai precisar construír unhas estruturas propias que lle permitan dar resposta ás limitacións expostas.

Na práctica, Sadakane implementou a súa solución dun xeito distinto a súa descripción teórica. Para resolver o feito de que non se ten A, e polo tanto non é suficiente determinar os intervalos $A[sp, ep]$ para localizar as ocorrencias e devolver os valores $A[i]$ no intervalo, tómanse mostraxes a intervalos regulares l do texto e almacénanse nun array $A_S[1, n/l]$ as posicións do array de sufixos que apuntan a esas mostraxes. Ademais márcanse estas posicións de A nun bitmap $B_A[1, n]$ que permite saber que se $B_A[i] = 1$ entón $A[i] = A_S[rank(B_A, i)]$. Se a posición non está marcada hai que probar $i \leftarrow \Psi(i)$ sucesivamente movéndose virtualmente en T unha posición en cada iteración. Se se determina que $A[i] = j$ despois de k accesos a Ψ, entón o valor orixinal era $j - k$. Grazas á mostraxe regular en T só se necesitan como moito l iteracións até encontrar unha posición das seleccionadas en A.

Por último, para poder descartar T, é necesario poder extraer calquera subcadea $T[a, b]$. Para as posicións do texto seleccionadas na mostraxe $j \cdot l$ almacénase $A^{-1}[j \cdot l]$ nun array $A_S^{-1}[1, n/l]$ na orde na que aparecen no texto. De esta forma é posible encontrar as últimas posicións da mostraxe $j \cdot l$ que preceden a a, $j = \lfloor a/l \rfloor$, e saber que $j \cdot l$ é apuntado por $i = A_S^{-1}[j]$. Desde ese i úsase o mecanismo descrito para extraer a cadea usando D e Ψ para obter a subcadea $T[j \cdot l, b]$ que contén á que nos interesa.

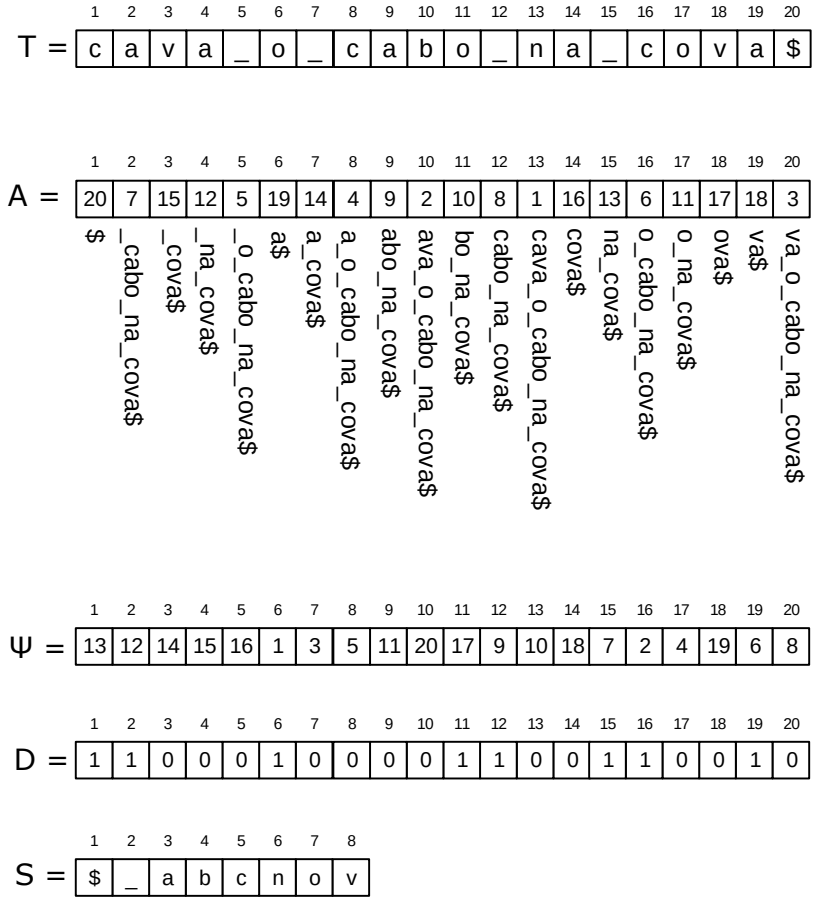

Figura 3.4: Autoíndice de Sadakane para o texto "`cava o cabo na cova$`".

Capítulo 4

Autoíndices orientados a palabras

Os autoíndices son estruturas que permiten representar un texto de forma comprimida e que, á vez, ofrecen funcionalidades eficientes de busca sobre el. Os primeiros autoíndices que aparecen na literatura consideran os textos a representar como unha secuencia de caracteres (autoíndice de texto completo), o que lles vai permitir ofrecer funcionalidades de busca de texto completo (*full-text search*), ou o que é o mesmo, atopar calquera cadea arbitraria de caracteres dentro do texto. Unha alternativa que permite reducir considerablemente o espazo ocupado polo autoíndice, cando se constrúe sobre texto en linguaxe natural, é considerar o texto como unha secuencia de palabras en vez de caracteres (autoíndice de palabras), mais a cambio, xa non será posible realizar buscas de texto completo. Isto non é demasiado significativo se temos en conta que a maior parte das buscas sobre texto en linguaxe natural van ser de palabras ou frases, ás que o autoíndice de palabras si vai poder dar resposta.

Neste capítulo preséntanse dúas variantes de autoíndices orientados a palabras, que seguen a idea proposta por Sadakane no seu autoíndice de texto completo para convertir o array de sufixos, neste caso orientado a palabras, nun autoíndice, e poder prescindir así do texto orixinal para operar. Como resultado, veremos que os autoíndices construídos van ocupar un espazo similar ao dos mellores compresores baseados en palabras, e resultan tamén máis rápidos para a busca de frases que os índices invertidos que ocupan un espazo equivalente.

O capítulo organízase da seguinte maneira. Na sección 4.1 profundamos un pouco máis na motivación desta investigación. Posteriormente nas seccións 4.2 e 4.3 detallamos as particularidades das dúas propostas presentadas, o `WCSA`, un autoíndice

simple baseado en `CSA` e orientado a palabras, e o `FWCSA`, a súa versión flexible con capa de presentación. Finalmente na sección 4.4 incluímos a avaliación experimental das novas estruturas.

4.1. Motivación

A idea de usar un autoíndice que empregue as palabras, en lugar dos caracteres, como elementos constituíntes dun texto en linguaxe natural semella prometedora, xa que un autoíndice que obteña entropía de orde superior debería ser quen de capturar a dependencia entre palabras consecutivas, que no caso da linguaxe natural é significativa [BCW90, capítulo 4]. Ademais, é destacable que mesmo un autoíndice lento como o `CSA`, é capaz de atopar as aparicións de frases de m palabras en tempo $O(m \log n + occ \cdot \log^{1+\epsilon} n)$, sendo occ as aparicións do patrón buscado e n a lonxitude to texto, e de saber o número de aparicións en só $O(m \log n)$. Estes resultados son mellores que os obtidos polos índices invertidos, que necesitan realizar intersección s de listas. Por exemplo, para unha frase de dúas palabras que aparecen occ_1 e occ_2 veces individualmente, un índice invertido tardaría $O(occ_1 + occ_2)$ ou $O(occ_1 \log occ_2)$, onde tanto occ_1 como occ_2 son (posiblemente moito) máis grandes que occ.

Usar un autoíndice sobre palabras en linguaxe natural xera novos desafíos. O primeiro é que o alfabeto é moi grande, o que impide o uso dos mellores esquemas teóricos [GGV03,FMMN07] que obteñen entropía de orden k a costa de $\Omega(\sigma^k)$ espazo extra, onde σ é o tamaño do vocabulario. É sabido que un texto de n palabras ten un vocabulario de tamaño $\sigma = O(n^\beta)$ [Hea78], onde $\beta \approx 0{,}5$ [BYRN99]. Polo tanto, σ^k pode ser $\Omega(n)$ incluso para valores pequenos de k. Non obstante, outros autoíndices como o `CSA` de Sadakane [Sad03] acércanse á entropía de orde superior sen esa dependencia de σ. A nosa primeira proposta, o `CSA` orientado a palabras (Word-Based CSA, `WCSA`), parte da consideración do texto como unha secuencia de identificadores de palabras e separadores e a súa representación cun `CSA`.

O segundo desafío é permitir unha maior flexibilidade nas buscas que a que ofrece unha busca estritamente literal. Por exemplo, os índices invertidos permiten normalmente buscar frases sen ter en conta se as palabras están separadas por un espazo, por dous, un tabulador, unha nova liña etc. Isto non é trivial co modelo do `WCSA` simple, onde vai ser preciso almacenar algunha información sobre os separadores para poder recrear exactamente o texto orixinal. Ademais, é común aplicar algún tipo de filtrado sobre as palabras do texto que queren ser buscadas [BYRN99]. Así, por exemplo, sería posible considerar as aparicións de `"preproceso"`, `"pre-proceso"`, e `"PRE-PROCESO"` como resultados da busca do termo `"preproceso"`, ou incluso as de `"preprocesando"` e `"preprocesado"`, obtidas tras un proceso de *stemming* que permite obter as raíces das palabras para a súa posterior indexación. Tamén é habitual descartar os *stopwords* (artigos, preposicións etc.) nas buscas. Polo tanto semella que

é necesaria unha *capa de presentación* onde o texto é filtrado deixando, dunha parte, unha *secuencia buscable* de palabras "desnudas" (posiblemente sen *stopwords*, en minúsculas e só coa raíz[1]), e, doutra banda, unha *secuencia de presentación* que vai conter os separadores e toda a información extra das palabras desnudas, necesaria para poder recuperar o texto orixinal. Así, só a secuencia buscable é autoindexada, mentres que a secuencia de presentación vai ser simplemente comprimida cunha técnica que permite acceso directo rápido para soportar a recuperación do texto orixinal de forma eficiente. Chamamos `WCSA` flexible (*Flexible WCSA*, `FWCSA`) a esta segunda estrutura.

Debemos facer notar que o `(F)WCSA`[2] está deseñado para traballar en memoria principal e, polo tanto, require que o tamaño do texto comprimido non exceda da memoria RAM dispoñible. Mentres que os índices invertidos obteñen bos resultados en memoria secundaria, o `(F)WCSA` ten uns patróns de acceso que fan pensar que non terá un bo desempeño nese escenario. Con todo, recentemente aumentou o interese dos índices invertidos que traballan en memoria principal [SC07, ST07], motivado principalmente polo abaratamento das memorias RAM de gran tamaño e polas arquitecturas distribuídas onde as coleccións de textos poden residir na memoria principal de varios ordenadores. Polo tanto, as estruturas de datos pensadas para memoria principal son de interese na actualidade, a diferenza do que se supoñía fai quince anos.

4.2. CSA orientado a palabras (Word-Based CSA)

Nesta sección presentamos o `CSA` simple orientado a palabras (Word-Based CSA, `WCSA`). Pódese considerar coma unha adaptación do `CSA` de Sadakane [Sad03] para un alfabeto de gran tamaño.

Para crear o `WCSA` debemos en primeiro lugar asignar un enteiro *id* a cada palabra ou separador diferente do texto fonte[3]. Posteriormente créanse *Sid*, a secuencia de enteiros composta polos identificadores das palabras do texto, e V, un array co vocabulario onde se almacenan as palabras correspondentes a cada *id*. Finalmente, *Sid* autoindéxase utilizando un `CSA` orientado a enteiros (`iCSA`). O algoritmo para crear o `iCSA` constrúe en primeiro lugar o array de sufixos A de *Sid* e D, de xeito que descarta *Sid*. Despois créanse os arrays A^{-1} e ψ, xunto con B_A, A_S e A_S^{-1}, tal e como foron descritos ao final da sección 3.4. Nese momento tanto A como A^{-1}

[1] A decisión sobre que é buscable e que non é buscable vai depender só das necesidades do usuario.

[2] Empregamos este nome para referirnos a ambas as dúas versións do autoíndice de palabras, `WCSA` e `FWCSA`.

[3] Procesamos o texto usando o modelo sen espazos: se unha palabra está seguida dun único espazo en blanco ese separador non é codificado pero inclúese de forma implícita durante en tempo de extracción do texto ou na descompresión. Isto aforra o 70 % dos separadores.

poden ser descartados. Supoñendo que existen σ palabras diferentes, o vocabulario usado por `iCSA` é $\{1, 2, \ldots, \sigma\}$, de xeito que queda implícito e non hai necesidade de almacenalo (e tampouco $S[1, \sigma']$). Finalmente, comprímese ψ almacenando mostras absolutas e utilizando Huffman para codificar as diferencias relativas, incluíndo unha codificación especial para os *runs*[4]. En resumo, `WCSA` consta dun vector de palabras V ordenado alfabeticamente e dunha capa de baixo nivel composta por un `iCSA` construído sobre Sid.

4.2.1. Operacións de busca no WCSA

Coma calquera autoíndice, o `iCSA` proporciona as seguintes funcións básicas usando os algoritmos descritos para o `CSA`: *countiCSA(P')* conta o número de aparicións do patrón P' en Sid; *locateiCSA(P')* localiza as posicións de P' en Sid; e *extractiCSA(l,r)* recupera os enteiros $Sid[l] \ldots Sid[r]$.

A busca dun patrón $P = \{p_1, p_2, \ldots, p_m\}$ no `WCSA` comeza cunha busca binaria en V de cada palabra p_i de P para obter o seu correspondente id_i, é dicir, a súa posición en V. Desta maneira obtense o novo patrón $P' = \{id_1, id_2, \ldots, id_m\}$ que debe ser buscado no `iCSA`. A operación *countWordsWCSA*(P) tradúcese directamente a *countiCSA*(P'), e *locateWordsWCSA*(P) a *locateiCSA*(P'), tendo en conta que obtén as posicións relativas de palabras, non de bytes, dos resultados da busca. Por último, *extractWordsWCSA*(s, e) recupera o texto orixinal desde a s-ésima á e-ésima palabra: obtéñense os ids das palabras con *extractiCSA*(s, e) e despois recupéranse as palabras orixinais almacenadas nesas posicións (ids) no array V. Pódense obter fragmentos de texto compostos por k palabras ao redor das aparicións de P' aplicando $occs =$*locateWordsWCSA*(P') seguido de *extractWordsWCSA*$(occs[i-k], occs[i+k])$ para cada $i \in [1 \ldots |occs|]$.

4.2.2. Representación compacta de Ψ

Esta sección presenta distintas técnicas para a representación compacta de Ψ. A obtención dunha representación de Ψ eficiente, tanto en espazo como en tempo de acceso, vai resultar crítica para a eficiencia xeral do autoíndice.

Inicialmente adaptamos as ideas da implementación de Ψ de Sadakane [Sad03, NM07] a alfabetos grandes. Así, gárdanse mostras absolutas a posicións $k \cdot t_\Psi$, $0 \leq k \leq \lfloor n/t_\Psi \rfloor$ nun array almacenado de forma compacta. Para cada mostra gárdase o valor absoluto de Ψ nesa entrada e un punteiro á secuencia de bits comprimida de Ψ onde as seguintes celas se almacenan de forma compacta e diferencial. Eses

[4] Estudáronse varias alternativas para a compresión de ψ, tal e como se mostran ao final na sección 4.2.2.

enteiros usan o número máximo de bits necesarios para representar todos os valores almacenados.

Na aproximación de Sadakane úsanse códigos γ para comprimir as diferenzas, e cando unha diferenza é igual a 1, o seguinte número codifica a lonxitude dese *run*, é dicir, o número de 1s consecutivos que seguen. Exploramos tamén outras alternativas, como os códigos δ, Huffman e algunhas combinacións de códigos.

Unha diferenza importante con respecto a alfabetos pequenos é que, na aproximación orientada a palabras, a secuencia diferencial $\Psi(i)-\Psi(i-1)$ pode conter un número significativo de valores negativos, até $\sigma - 1$. Mentres que nun alfabeto de caracteres este valor é 255, nun alfabeto de palabras pode ser moito maior. Por exemplo, o noso texto de 1GB (o texto descrito na sección 4.4) produce ao redor de 228 millóns de palabras. Desas 228 millóns de diferenzas en Ψ, aproximadamente medio millón corresponden a valores negativos. Se ben é certo que non é unha porcentaxe elevada, é necesario codificar estes valores axeitadamente para non afectar negativamente á compresión.

Cando usamos códigos δ debemos codificar os valores negativos dalgunha maneira como excepcións. Tamén consideramos unha aproximación tradicional onde, no canto de representar a diferenza que pode ser positiva ou negativa, representamos con códigos δ o valor $x_i = \Psi(i)\ xor\ \Psi(i-1)$[5]. Nótese que desta forma x_i sempre é positivo, e se $x_1 = \Psi(1)$ entón $\Psi(i)$ pode obterse como $x_i\ xor\ \Psi(i-1)$. Chamaremos δ-*dif* a variante que representa diferenzas e δ-*xor* a que representa *xor*. De maneira análoga definimos γ-*dif* e γ-*xor*.

A codificación Huffman parece en principio mellor que as codificacións anteriores en termos de espazo, mais como existen moitos valores diferentes, o tamaño do modelo supón un problema. Na práctica, a codificación Huffman debe ser combinada con algunha das codificacións anteriores. Ademais tamén introducimos de diferentes formas a codificación *run-length* para as secuencias de 1s, o que permite capturar as dependencias entre palabras na compresión.

Desenvolvemos doce alternativas diferentes para representar os valores non correspondentes a mostras de Ψ. Estas variantes permítennos estudar o efecto da codificación dos *runs* de 1s en Ψ, a codificación diferencial fronte a codificación de *xor* de valores consecutivos, e o impacto de usar os códigos Huffman ou δ en lugar dos códigos γ tal e como se facía no `CSA` orixinal.

- *Huffman-rle*. Esta estratexia baséase na combinación da codificación Huffman coa codificación de *runs* en Ψ. Usamos dúas codificacións diferentes de Huffman: unha codificación principal H^c para representar os valores de Ψ e unha codificación secundaria H^r para representar as lonxitudes dos *runs* dentro de

[5] *xor* refírese ao *eXclusive OR*, representado con '^' en `C/C++`.

Ψ. A representación *Huffman-rle* contén un código Huffman (de H^c) por cada valor que non é mostra $\Psi(i)$ (excepto para posicións adxacentes en Ψ que se codifican cun *run* de 1s). Cando se representan estes valores de $\Psi(i)$ preséntanse tres casos diferentes que se tratan dependendo do valor $d_i = \Psi(i) - \Psi(i-1)$: *i) Diferenzas pequenas* son codificadas directamente cun código Huffman $H^c_{d_i}$. O número de *valores pequenos* configúrase cun parámetro s: as diferenzas d_i tales que $2 \leq d_i < s$ codifícanse co seu propio código Huffman ($H^c_2 \dots H^c_{s-1}$). *ii) Diferenzas negativas ou grandes.* Cando ben $d_i < 0$ ou $d_i \geq s$, utilízase un código de escape (H^c_0) seguido do número $\Psi(i)$ representado con $\lceil \log_2 n \rceil$ bits. *iii) Run-encoding.* Se $d_i = 1$, detéctase un *run* de 1s en $\Psi(i)$. Supoñendo que ten lonxitude l, o *run* codifícase emitindo primeiro un código de escape H^c_1 e representando despois o valor de l. Como l adoita ser moi pequeno ($1 \leq l < t_\Psi$, xa que os *runs* córtanse artificialmente ao alcanzar o valor da seguinte mostra), utilízase unha codificación de Huffman secundaria H^r para representar os valores l.

Polo tanto, a codificación Huffman H^c obtense da seguinte maneira. En primeiro lugar considérase un vocabulario composto dos símbolos $\{0 \dots s-1\}$. Calcúlase a frecuencia de cada símbolo de forma que $frec[i]$, $i \in 2 \dots s-1$ conta o número de veces que ocorre o valor de diferenza i. Engadimos a $frec[1]$ o número de *runs* que existen en Ψ e a $frec[0]$ o número de valores grandes e negativos. Despois ordenamos os símbolos por frecuencia e aplicamos o algoritmo de Huffman para obter o código H^c_i que corresponde a cada símbolo. De forma similar obtense H^r, supoñendo un vocabulario composto polos símbolos $l \in 1 \dots t_\Psi - 1$ e contando o número de veces que existe un *run* de lonxitude l

Para comprobar o efecto de non codificar os *runs* desenvolvemos unha versión simplificada de *Huffman-rle* que no usa codificación *run-length*. Así, cada aparición dunha diferenza $+1$ represéntase cun código H^c_1. Chamamos a esta variante *Huffman*.

- *Huffman-rle-opt.* Baseándonos na proposta anterior deseñamos unha técnica mellorada para comprimir Ψ que aborda as súas dúas maiores debilidades: a necesidade dun código de escape (H^c_1) antes dun *run* e a necesidade de representar $\Psi(i)$ con $\lceil \log_2 n \rceil$ bits despois dun código de escape H^c_0 no caso de diferenzas negativas ou grandes. Dos s símbolos que se poden codificar ca codificación Huffman H^c reservamos $w + w$ símbolos, onde w é o tamaño de palabra da máquina, para representar a lonxitude da representación binaria das diferenzas negativas e grandes. Ademais resérvanse t_Ψ símbolos para codificar a lonxitude dos *runs* de 1s, e os restantes $s - 2w - t_\Psi$ símbolos utilízanse para representar as diferenzas pequenas.

  Desta forma temos catro casos diferentes: *i)* Os códigos $H^c_2 \dots H^c_{runini-1}$, onde $runini = s - 2w - t_\Psi$, codifican directamente diferenzas $d_i = \Psi(i) - \Psi(i-1)$ que cumpren $2 \leq d_i < runini$. *ii)* Os seguintes t_Ψ códigos,

$H^c_{runini} \dots H^c_{posini-1}$, onde $posini = runini + t_\Psi$, codifican directamente *runs*. Máis concretamente, un *run* de lonxitude l codifícase como $H^c_{runini+l}$. *iii)* Os códigos $H^c_{posini} \dots H^c_{negini-1}$, onde $negini = posini + w$, úsanse como códigos de escape para introducir diferenzas grandes d_i, e implicitamente representan a lonxitude da representación binaria de d_i. Supoñendo $l = \lceil \log_2 d_i \rceil$, $\Psi(i)$ codifícase co código $H^c_{posini+l}$ seguido pola representación binaria de d_i usando só l bits. *iv)* Analogamente, os códigos $H^c_{negini} \dots H^c_{s-1}$ resérvanse para os valores negativos. Deste xeito gárdase de forma implícita o signo de d_i e procédese como no caso dos valores grandes, neste caso representando $-d_i$. Supoñendo $l = \lceil \log_2(-d_i) \rceil$, $\Psi(i)$ codifícase co código $H^c_{negini+l}$ seguido de l bits que representan $-d_i$ en binario.

Como na aproximación anterior, implementamos unha versión simplificada de *Huffman-rle-opt*, chamada *Huffman-opt*, que non codifica *runs*.

- *δ-dif* e *γ-dif*. Como os códigos δ só poden representar números positivos, reservamos algúns valores positivos (1 valor por cada K enteiros positivos) para representar os valores negativos. O parámetro K configúrase para maximizar a compresión. Basicamente unha diferenza negativa d_i represéntase como o código δ de $-Kd_i$, mentres que un valor positivo d_i codifícase como o código δ de $(Kd_i - 1)/(K - 1)$. Por exemplo, se $K = 3$, 1 represéntase con código δ de 1, 2 como o código δ de 2, 3 como o código δ de 4, 4 como o código δ de 5, 5 como o código δ de 7 etc; mentres que -1 represéntase co código δ de 3, -2 como o código δ de 6, e así sucesivamente.

  Para a descodificación dun valor C, se $v \leftarrow \delta\text{-decode}(C)$, entón devólvese $-v/K$ se $v \ mod \ K = 0$ ou $v - (v/K)$ en caso contrario. Coa codificación γ o proceso é análogo.

  Creamos tamén variantes *δ-dif-rle* e *γ-dif-rle* que sacan partido da codificación de *runs* de lonxitude l. De novo, a idea é que cando se codifica un *run* se utiliza o valor $+1$ como código escape seguido da codificación de l.

- *δ-xor* e *γ-xor*. En lugar das codificacións de diferenzas da representación anterior, neste caso represéntanse os valores *xor* cos valores anteriores de Ψ, como foi comentado anteriormente. Desta forma, utilízanse os códigos δ ou γ de $x_i = \Psi(i) \ xor \ \Psi(i-1)$. Para descodificar simplemente $\Psi(i) \leftarrow x_i \ xor \ \Psi(i-1)$.

  A versión que inclúe a codificación de *runs*, é dicir, *δ-xor-rle*, require, como no caso de *δ-dif-rle*, o uso do valor $+1$ como código escape e a codificación posterior da lonxitude do *run*. Con todo, como se reserva o valor $+1$, a codificación de valores normais $x_i = \Psi(i) \ xor \ \Psi(i-1)$ require un truco adicional, xa que podería suceder que $\Psi(i) \ xor \ \Psi(i-1) = 1$ aínda cando $\Psi(i) \neq \Psi(i-1)+1$ (e polo tanto non é un *run*). Por exemplo, se $\Psi_i = 8$ e $\Psi_{i-1} = 9$ obtemos que $\Psi_i - \Psi_{i-1} = -1$, pero (en binario) $1000 \ xor \ 1001 = 0001$. Para resolver este problema codificamos $x'_i = (\Psi(i) \ xor \ \Psi(i-1)) + 1$ en vez de $x_i = \Psi(i) \ xor \ \Psi(i-1)$. A

descodificación realízase como $\Psi(i) = (x'_i - 1)\ xor\ \Psi(i-1)$. γ-*xor-rle* obtense de forma similar.

A figura 4.1 compara as diferentes variantes para comprimir Ψ para un texto dun 1 GB, onde a representación plana de Ψ require ao redor de 872MB. Mostramos o tamaño da representación comprimida de Ψ e o tempo necesario para acceder a todos os valores de Ψ, comezando na posición $i \leftarrow 0$ até a posición $i \leftarrow \Psi(i)$ de forma repetida con todos os posibles valores i (é dicir, percorrendo o texto de esquerda a dereita). Para os códigos Huffman usamos $s = 2^{14}$, que permite representar a maior parte dos valores interesantes con un modelo de tamaño moderado. Usamos potencias de 2 para o período de mostraxe t_Ψ, é dicir, 1024, 512, 256, e sucesivamente. Nas gráficas mostramos o caso $t_\Psi = 16$, e despois cada punto hacia a esquerda representa a seguinte potencia de 2 (valores máis grandes de t_Ψ implican menos espazo e máis tempo).

A figura 4.1-superior-esquerda mostra que δ-*dif* supera claramente a δ-*xor* e que δ-*dif-rle* é capaz de obter unha pequena mellora sobre δ-*dif*. Isto cúmprese tamén para as variantes que usan códigos γ, que son superadas claramente polas que usan códigos δ. A única excepción é γ-*xor*, que obtén os mellores resultados para mostraxes moi densas (obtendo porcentaxes de compresión peores que o 50 %). A figura 4.1-superior-dereita mostra esa parte da gráfica en detalle.

A figura 4.1-inferior compara as técnicas baseadas en Huffman con aquelas que usan códigos δ. A gráfica inferior-dereita inclúe unha área ampliada da gráfica inferior-esquerda. Pode observarse que a codificación δ é moi competitiva cando se usa unha mostraxe densa de Ψ ($t_\Psi \leq 32$), que xera un índice pesado pero rápido. Por exemplo, δ-*dif-rle* con $t_\Psi = 64$ obtén uns resultados espazo-temporais lixeiramente mellores que os de *Huffman-rle-opt* con $t_\Psi = 32$. Non obstante, cunha mostraxe máis dispersa, as técnicas *Huffman-rle* obteñen mellores resultados.

Nos resultados experimentais da sección 4.4 úsase a alternativa *Huffman-rle-opt*, con $s = 2^{14}$, para a compresión de Ψ.

4.3. Flexible Word-Based CSA

Presentamos agora un índice máis flexible baseado no `WCSA`. O `WCSA` flexible (*Flexible* ***`WCSA`***, `FWCSA`) pode soportar moitos requerimentos típicos das buscas en linguaxe natural, como buscas sen distinción de maiúsculas ou minúsculas, busca de raíces de palabras tras un proceso de *stemming*, omisión de *stopwords* e/ou separadores etc. O `FWCSA` non indexa exactamente o texto orixinal, senón unha versión *normalizada* do mesmo. A normalización é unha función definida polo usuario de palabras e separadores do texto orixinal a palabras normalizadas ou

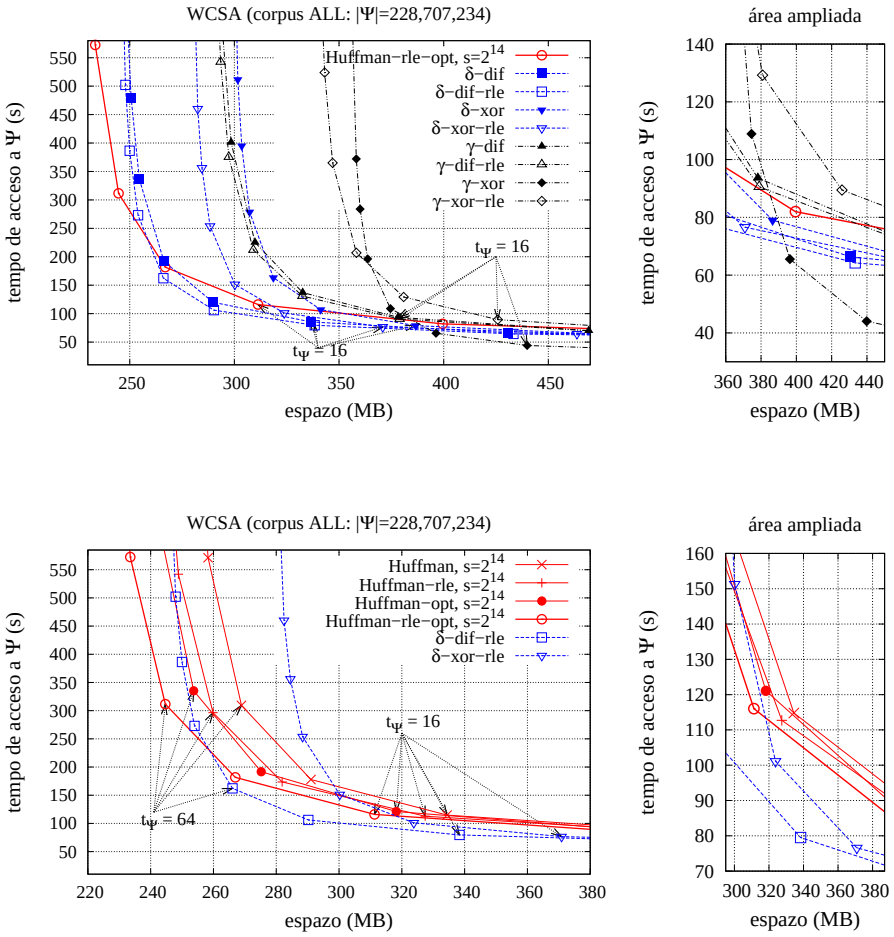

Figura 4.1: Comparativa dos resultados espazo-temporais obtidos por diferentes representacións comprimidas de Ψ.

á palabra baleira. Permite expresar os requerimentos comentados anteriormente[6]. Facemos corresponder o conxunto de palabras normalizadas diferentes cos enteiros *ids* e substituímos cada palabra do texto orixinal polo *id* da súa versión normalizada (ou ignorámola se a normalización correspóndese coa palabra baleira), e finalmente construímos o `iCSA` para a secuencia de *ids* resultante.

Como queremos que o `FWCSA` sexa capaz de recuperar calquera fragmento do texto orixinal necesitamos almacenar algunha información adicional na que denominamos *capa de presentación*.

A figura 4.2 mostra a estrutura xeral do `FWCSA`. É necesaria unha primeira pasada sobre o texto orixinal para recompilar algunhas estatísticas do texto fonte. Dividimos o texto orixinal en "palabras válidas" e "separadores". Unha "palabra válida" é unha palabra do texto ou separador[7] ás que a normalización non asigna a unha palabra baleira. En este contexto, un "separador" é a secuencia do texto que hai entre dúas palabras válidas, é dicir, a secuencia maximal de palabras de texto e separadores asignadas á palabra baleira pola normalización. Polo tanto no texto altérnanse sempre palabras válidas e separadores[8]. Créase entón un vocabulario de palabras *canónicas* (é dicir, normalizadas) e almacénase ordenado alfabeticamente. Para cada palabra canónica almacénase unha lista ordenada por frecuencia con todas as variantes que o proceso de normalización asignou a esa palabra. Analogamente créase un vocabulario con todos os "separadores" do texto fonte e ordénase por frecuencia.

Unha segunda pasada sobre o texto orixinal permite encher as estruturas da capa de presentación, tal e como ilustra a figura 4.2, así como o array *Sid*. No exemplo $Sid[1] = 2$ porque a primeira palabra válida do texto, "`Blue`", asígnase coa normalización á segunda palabra canónica, "`blue`". Unha vez rematada a construción da capa de presentación constrúese a estrutura do `iCSA` sobre a secuencia *Sid*.

Na capa de presentación existe un bitmap CT que garda a representación comprimida do aspecto de presentación do texto. Baseándose na alternancia de palabras e separadores, CT conterá o código correspondente dunha palabra seguido do código dun separador e así sucesivamente. Como exemplo, na figura 4.2 podemos observar que $CT[1 \ldots 3] =$'001' é o código asociado ao separador "`The `" e $CT[4] =$'1' é o código da variante "`Blue`" da palabra canónica "`blue`". Estes códigos obtéñense da seguinte maneira: dunha banda asígnase aos separadores un código usando o algoritmo Huffman orientado a palabras [Mof89, Huf52] sobre o vocabulario completo

[6] Por exemplo se queremos buscas que non distingan maiúsculas de minúsculas ignorando *stopwords* e separadores, a normalización podría asignar a todas as palabras a súa versión en minúsculas e aos *stopwords* e separadores a palabra baleira.

[7] O que é unha palabra tamén pode ser definido polo usuario, sendo a definición típica a secuencia maximal de letras e díxitos.

[8] Se a normalización desexase almacenar separadores como palabras válidas podemos inserir "separadores" ficticios entre as palabras válidas.

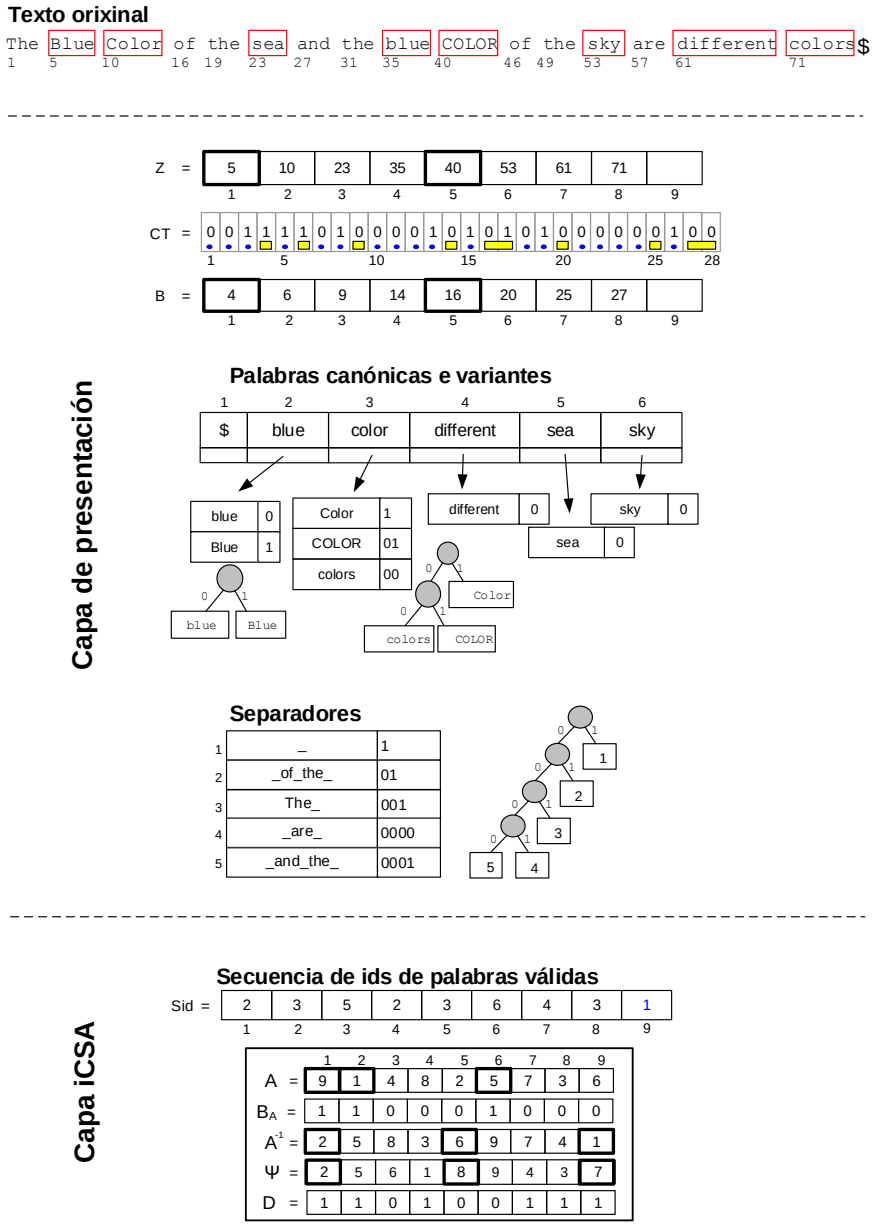

Figura 4.2: Estrutura xeral do FWCSA.

de separadores (o almacenamento da forma da árbore require un sobrecusto pequeno cando se usa Huffman canónico [MK95]). Por outra banda as variantes de cada palabra canónica, que están tamén gardadas ordenadas por frecuencia, codifícanse co mesmo método. Polo tanto, para a descodificación é necesario coñecer, ademais das variantes de cada palabra canónica, a forma da árbore de Huffman usada na codificación. Na práctica cando unha palabra canónica ten só unha única variante realmente non se codifica en CT (aínda que no exemplo da figura 4.2 usamos 1 bit por motivos de claridade). Así, xunto coa información sobre as palabras canónicas proporcionada por Sid (que non se almacena explicitamente pero pode obterse usando o `iCSA`), podemos recrear o texto orixinal, xa que Sid indica que árbore de Huffman hai que acceder durante a descodificación de palabras de CT.

Para posibilitar a descodificación desde calquera posición de palabra aleatoria do texto engadimos sincronismo aos códigos de CT usando un vector B. Dada unha posición i en Sid, $B[i] = p$ indica a posición en CT desde onde se pode descodificar a correspondente variante da palabra canónica $j = Sid[i]$ (usando a árbore de Huffman asociada á j-ésima palabra canónica). Despois de descodificar un símbolo desde ese punto p en CT encontraremos o inicio do código dun separador, despois o código da variante da palabra canónica $Sid[i + 1]$ e así sucesivamente. No noso exemplo podemos ver que $B[5] = 16$ é o inicio en CT do código '01' que corresponde coa terceira palabra canónica ($Sid[5] = 3$) ('01' → "`COLOR`"). Logo, $CT[18, 19]$ ='01' é o código do separador " `of the` ".

Necesítase un segundo array Z para as operacións *locate* e *display*. Este array fai corresponder cada posición i do vector Sid á súa posición real en bytes no texto orixinal T: $Z[i] = j$ significa que $T[j]$ é o primeiro carácter da palabra representada por $Sid[i]$.

Para aforrar espazo tómanse mostras tanto en B como en Z en posicións regulares $i \cdot k_b$ e $i \cdot k_z$, respectivamente, e almacénanse só esas posicións. Pódese obter un valor p de B aínda que non sexa unha das mostras, onde $i \cdot k_b < p < i \cdot (k_b + 1)$, movéndonos á posición $B[i \cdot k_b]$ en CT e descodificando $p - i \cdot k_b$ palabras e separadores de forma alternativa. O número de bits descodificados en CT sumado ao valor $B[i \cdot k_b]$ indícanos o valor de $B[p]$. De forma similar pódese obter un valor p de Z engadindo o valor da mostra anterior $Z[i \cdot k_z]$ o número de caracteres descodificados despois de procesar $p - i \cdot k_z$ palabras e $p - i \cdot k_z$ separadores. Neste caso a descodificación empeza na posición $B[i \cdot k_z]$ de CT.

4.3.1. Estruturas da capa de presentación

Amosamos nesta sección os detalles máis importantes das estruturas utilizadas para a almacenaxe do vector de palabras canónicas, coas súas variantes e árbores de Huffman, así como dos separadores.

Array de palabras canónicas. Cada vez que necesitamos encontrar un *id* asociado a unha palabra canónica debe realizarse unha busca binaria, comparando cadeas. Como se amosa na figura 4.3.a), as palabras canónicas gárdanse en formato plano nun vector *bufferCanónicas*, soportando acceso directo grazas a un array *canónicas* con entradas de k bits, con $k = \lceil \log_2(|bufferCanónicas|) \rceil$. Esta estrutura sinxela reduce o sobrecusto do uso dun array de palabras baseado en punteros, baixando de 32 a 20–22 bits por símbolo. Ademais permite obter a lonxitude da i-ésima palabra canónica, xa que $len(i) = canónicas[i+1] - canónicas[i]$.

Variantes das palabras canónicas. As variantes das palabras canónicas só se utilizan cando se descodifica o texto (cando se mostra - *display* - ou se extrae - *extract*). Pódese reducir o seu tamaño codificándoas cun Huffman orientado a carácter en vez de mantelas en formato plano. A figura 4.3.c) mostra unha representación en tres niveis que soporta acceso en tempo constante ao código asociado ao primeiro carácter de calquera variante. O terceiro nivel almacena as variantes comprimidas. O segundo nivel (*SubDir*) garda os valores absolutos e saltos relativos á primeira e o resto de variantes dunha palabra canónica. Todos os enteiros da figura codifícanse usando $\lceil \log(max) \rceil$ bits para reducir o espazo. Supoñendo que unha mostra absoluta de *SubDir* usa M bits e un salto é codificado con J bits, o acceso ao código Huffman da j-ésima variante da i-ésima palabra canónica implica: *i)* ler o valor *absoluto* na posición $p = Dir[i]$ de *SubDir*; *ii)* se $j > 1$, ler o valor do *salto* na posición $Dir[i] + M + (j-2)J$; e *iii)* calcular $offset = absoluto + salto$.

Árbore de Huffman das variantes. A almacenaxe da forma da árbore canónica de Huffman require só gardar a súa altura H e dous vectores de H elementos que denominamos *num* (número de elementos en cada nivel) e *fst* (primeiro código en cada nivel) [MK95], o que supón normalmente un sobrecusto pequeno. Con todo, é necesario almacenar a forma dunha árbore canónica para cada palabra canónica (a árbore usada para codificar as súas variantes). A figura 4.3.d) mostra unha forma compacta de gardar todas esas árbores de Huffman que soporta acceso en tempo constante a calquera valor de *num* ou *fst* para cada palabra canónica. Na práctica estes vectores conteñen moi poucos valores, xa que o número máximo de variantes dunha palabra canónica é pequeno.

Separadores. Os separadores almacénanse nunha estrutura semellante á usada para as palabras canónicas. A diferenza é que neste caso os separadores almacénanse comprimidos cun Huffman orientado a caracteres en lugar de en forma plana (como as palabras canónicas). A estrutura usada móstrase na figura 4.3.b).

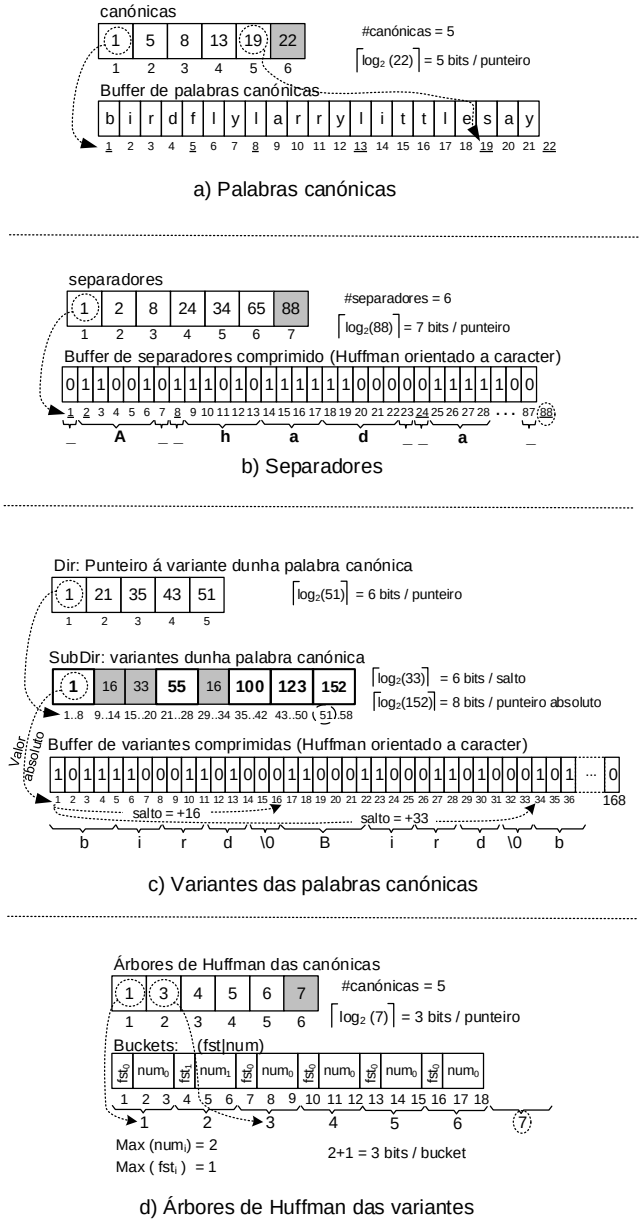

Figura 4.3: Estruturas compactas usadas na capa de presentación do `FWCSA`.

4.3.2. Operacións de busca

No `FWCSA` a execución das operacións empeza na capa de presentación e na maior parte dos casos é necesario acceder ao vector Sid a través das funcións $countiCSA(P')$, $locateiCSA(P')$, e $extractiCSA(left, right)$.

- $count(P)$: Contar o número de aparicións dun patrón (palabra individual ou frase) involucra realizar primeiro un paso de normalización de P para obter a secuencia das súas palabras válidas normalizadas. O novo patrón P' está formado pola secuencia dos *ids* desas palabras válidas (o que require buscas binarias no array de palabras canónicas). Finalmente, $count(P)$ calcúlase como $countiCSA(P')$, en tempo $O(m \log(n))$.
- $locate(P)$: Devolve as posicións do texto no que aparece o patrón P. Despois de crear o patrón P' como no caso anterior, $locateiCSA(P')$ devolve as posicións de palabra $i_1, \ldots, i_k$ de P' en Sid. Posteriormente calcúlanse as posicións en bytes no texto orixinal $Z[i_1] \ldots Z[i_k]$ usando os valores das mostras Z tal e como se explicou anteriormente.
- $extract(s, e)$: Recupera a subcadea $T_{s \ldots e}$ do texto orixinal. Realízase primeiro unha busca binaria en Z para obtener o último $Z[i] \leq s$. Posteriormente sincronizámonos co texto comprimido en CT movéndonos á posición $B[i]$. Desde alí descomprimimos usando CT até que o número de caracteres descomprimidos sexa maior que $e - Z[i]$. Os caracteres anteriores a s e posteriores a e son ignorados na resposta final.

Como no caso do `WCSA`, as funcións *locate* e *extract* pódense simplicar para que se refiran á palabras dentro do texto indexado en lugar de a caracteres do texto orixinal. Así pode evitarse a necesidade do array Z, e polo tanto aforrar espazo, ademais de mellorar o tempo de ambas as dúas operacións. Definimos tres operacións máis do `FWCSA` para este modelo: *locateWordsFWCSA*(), *snippetWordsFWCSA*() e *extractWordsFWCSA*().

- *locateWordsFWCSA*(P, r): Despois de obter o patrón filtrado P' de P, esta operación devolve o par $(o_{ct},\ o_{sid})$ para cada aparición de P': o_{ct} é a posición en CT correspondente a r palabras antes da aparición e o_{sid} é o índice correspondente en Sid. Cada par contén información suficiente para comezar a extracción dun fragmento de texto que empeza r palabras antes de cada aparición de P. A súa implementación dedúcese de maneira obvia da implementación de *locate*.
- *snippetWordsFWCSA*(P, r): Obtén o fragmento de texto composto por $2 \times r$ palabras válidas comezando r palabras antes que calquera aparición de P'.

Usando os pares obtenidos con *locateWordsFWCSA* esta operación simplemente descodifica as palabras e os separadores de forma alternada.

- *extractWordsWCSA*(s, e): Recupera o código orixinal que comeza na s-ésima palabra válida descomprimindo $e - s + 1$ palabras válidas e consecuentemente $e - s$ separadores.

4.4. Resultados experimentais

Usamos unha colección de texto grande de 1023MB obtenida coa agregación de varios corpus de TREC-2: AP Newswire 1988 (AP) e Ziff Data 1989–1990 (ZIFF), e tamén de TREC-4: Congressional Record 1993 (CR) e Financial Times 1991–1994, e finalmente o corpus Calgary[9]. A máquina utilizada nos experimentos é unha Intel®Pentium®-IV 3.00 GHz, con 4GB de RAM na que corría un Debian (kernel 2.4.27). Compilamos usando gcc versión 3.3.5 con optimizacións `-O9`. Os resultados de tempos medíronse usando o tempo de usuario de CPU.

Comparamos os nosos autoíndices `WCSA` e `FWCSA` con dous índices invertidos con direccionamento por bloques en memoria principal (`II` e `FII`) con propiedades semellantes[10]. `II` é o índice dos autores do traballo [BFLN08] e `FII` é a súa versión *Flexible*, onde o texto é comprimido con `ETDC` [BFNP07] e as listas codifícanse de forma diferencial con `ETDC`, gardando mostras absolutas cada k valores para acelerar as intersección. Esta aproximación difire levemente coa utilizada en [CM07] e obtén resultados semellantes na práctica. O proceso de normalización para `FWCSA` e `FII` consistiu en: (1) escoller como palabras válidas as secuencias alfanuméricas maximais, (2) omitir separadores e *stopwords*, e (3) usar minúsculas.

Medimos os tempos de localizar aparicións de patróns (*locateWords*) e os tempos necesarios para extraer fragmentos de 20 palabras ($r = 10$) ao redor das aparicións de patróns dados (*snippetWords*). Usamos 100 patróns de proba correspondentes a catro grupos diferentes de patróns de palabras individuais, con diferentes rangos de frecuencias, e catro grupos de patróns de frases compostas por 2, 4, 6 e 8 palabras. Os resultados tanto de localización como da extracción de fragmentos correspondénse ao tempo medio por aparición (en milisegundos por aparición).

[9] `http://corpus.canterbury.ac.nz/descriptions/`

[10] Probáronse algúns índices invertidos dispoñibles libremente pero eran: *i)* orientados a palabra completa, que ocupan moito máis espazo que `(F)WCSA`, como por exemplo *Zettair* (`http://www.seg.rmit.edu.au/zettair/`) e *Wumpus* (`http://www.wumpus-search.org`); *ii)* enfocados a modelos de recuperación de información alternativos, coma *Lemur* (`http://www.lemurproject.org`); ou *iii)* non eran públicos, non estaban preparados para ser usados ou non se puideron instalar, como *galago* (`http://www.lemurproject.org/galago.php` [SC07]), os propostos en [CM07] e [ST08], e *Terrier* (`http://ir.dcs.gla.ac.uk/terrier/`).

WCSA vs II

Consideramos dúas configuracións diferentes de uso de memoria para os índices. Para WCSA modificamos os parámetros da capa do iCSA variando os intervalos das mostraxes das súas estruturas: $\{t_\psi, t_A, t_{A^{-1}}\}$. A primeira configuración, chamada $\texttt{WCSA}_1$, usa $\{t_\psi, t_A, t_{A^{-1}}\} = \{16, 16, 64\}$; a outra, $\texttt{WCSA}_2$, configurouse con $\{32, 32, 64\}$. Para II modificáronse os valores dos seus parámetros $\{k, b\}$, que se refiren ao intervalo de mostraxes para a indexación das súas listas comprimidas e ao tamaño do bloque (en KB). Chamamos $\texttt{II}_1$ á configuración $\{k, b\} = \{8, 16\}$, e $\texttt{II}_2$ a $\{k, b\} = \{32, 256\}$.

O cadro 4.1 mostra como $\texttt{WCSA}_2$ supera a $\texttt{II}_2$ en todos os aspectos. Na práctica, cando a dispoñibilidade de memoria é pequena o WCSA é claramente a mellor opción. Só cando queremos usar moita máis memoria o $\texttt{II}_1$ pode competir con $\texttt{WCSA}_1$ na extracción de fragmentos de palabras individuais ou frases curtas. Con todo, $\texttt{WCSA}_1$ segue sendo máis rápido que $\texttt{II}_1$ para a localización de aparicións. Cando buscamos patróns de frases a diferenza no rendemento entre WCSA e II aumenta a medida que aumentamos o número de palabras na frase.

FWCSA vs FII

Usamos tres configuracións diferentes para FWCSA usando os valores fixos $B = 32$ e $Z = 512$ para a capa de presentación e variando os intervalos dos parámetros das mostraxes do seu iCSA. A primeira configuración, chamada $\texttt{FWCSA}_1$, usa os valores $\{16, 16, 32\}$; $\texttt{FWCSA}_2$ obténse cos valores $\{32, 16, 64\}$; e $\texttt{FWCSA}_3$ usa os valores $\{32, 32, 64\}$. Para FII, $\{k, b\}$ configúranse a {64,16} para obter $\texttt{FII}_1$; $\texttt{FII}_2$ créase cos valores $\{64, 128\}$; e finalmente $\texttt{II}_3$ usa os valores $\{64, 1024\}$.

Os resultados, incluídos no cadro 4.2, mostran que cando a porcentaxe de compresión é cercana ao 40 % non existe un gañador claro. FWCSA é mellor que FII para frases longas pero FII obtén os mellores resultados para palabras de alta frecuencia, mais cando a cantidade de memoria decrece os resultados de FII empeoran moito máis rápido que os de FWCSA.

Tamén é destacable que II e FII non poden comprimir máis que un 35 %, e nesas porcetaxes (F)WCSA obtén mellores resultados. Non só iso, senón que podemos configurar (F)WCSA para que ocupe un menor espazo (ao redor dun 30 %), aínda que cunha clara perda en rendemento.

Un traballo posterior á publicación da nosa proposta [TS10] inclúe comparacións adicionais do WCSA con índices invertidos en memoria principal. Usando unha mostraxe máis densa obteñen porcentaxes de compresión no rango 42 %–53 % e confirman os resultados aquí expostos, xa que o WCSA obtén resultados superiores aos índices invertidos para as buscas de frases.

Cadro 4.1: Resultados comparando WCSA con II.

		$\texttt{WCSA}_i$		$\texttt{II}_i$	
		i=1	i=2	i=1	i=2
	Ratio (%)	45,03	38,08	45,54	39,07
		Locate			
Palabras	1–100	0,009	0,018	0,018	0,246
(rango	101–1000	0,007	0,019	0,019	0,237
frec)	1001–10000	0,006	0,019	0,023	0,163
	10000+	0,006	0,019	0,014	0,029
	2	0,005	0,014	0,028	0,113
Frases	4	0,005	0,009	1,128	3,737
#palabras	6	0,069	0,069	14,028	76,319
	8	0,059	0,059	7,396	50,118
		Snippet			
Palabras	1–100	0,055	0,091	0,027	0,255
(rango	101–1000	0,053	0,083	0,021	0,238
frec)	1001–10000	0,054	0,084	0,024	0,164
	10000+	0,054	0,084	0,015	0,030
	2	0,046	0,070	0,028	0,114
Frases	4	0,029	0,043	1,130	3,737
#palabras	6	0,069	0,139	14,028	76,389
	8	0,118	0,118	7,396	50,059

Alternativas non baseadas en índices invertidos

Nos mesmos anos que presentamos o WCSA, e cunha filosofía semellante, apareceron na literatura outros competidores aos índices invertidos [BFLN08, FBN+12]. Nesta sección comparámonos con eles. Para eso incluímos, ampliamos e discutimos os resultados expostos en [BFLN12] onde se presenta o WTPH, un autoíndice baseado na estrutura de wavelet tree e orientado a palabras, que utiliza de base o codificador Plain Huffman. Neste traballo compárase o WTPH con dous índices invertidos e co WCSA presentado neste capítulo. Engadimos nesta comparativa o WSSA, outro autoíndice baseado en arrays de sufixos [FBN+12]. Todos os índices configúranse para obter diferentes porcentaxes de compresión variando os seus parámetros (os intervalos das mostraxes do WCSA, t_{pos}, t_{bit_1} e t_{bit_2} para WSSA, o tamaño dos bloques e superbloques no WTPH e os intervalos de mostraxes dos índices invertidos).

Como se poden ver nas figuras 4.4–4.7 os resultados mostran que na localización de palabras e frases curtas o WTPH é máis rápido que o WCSA, pero o WCSA claramente

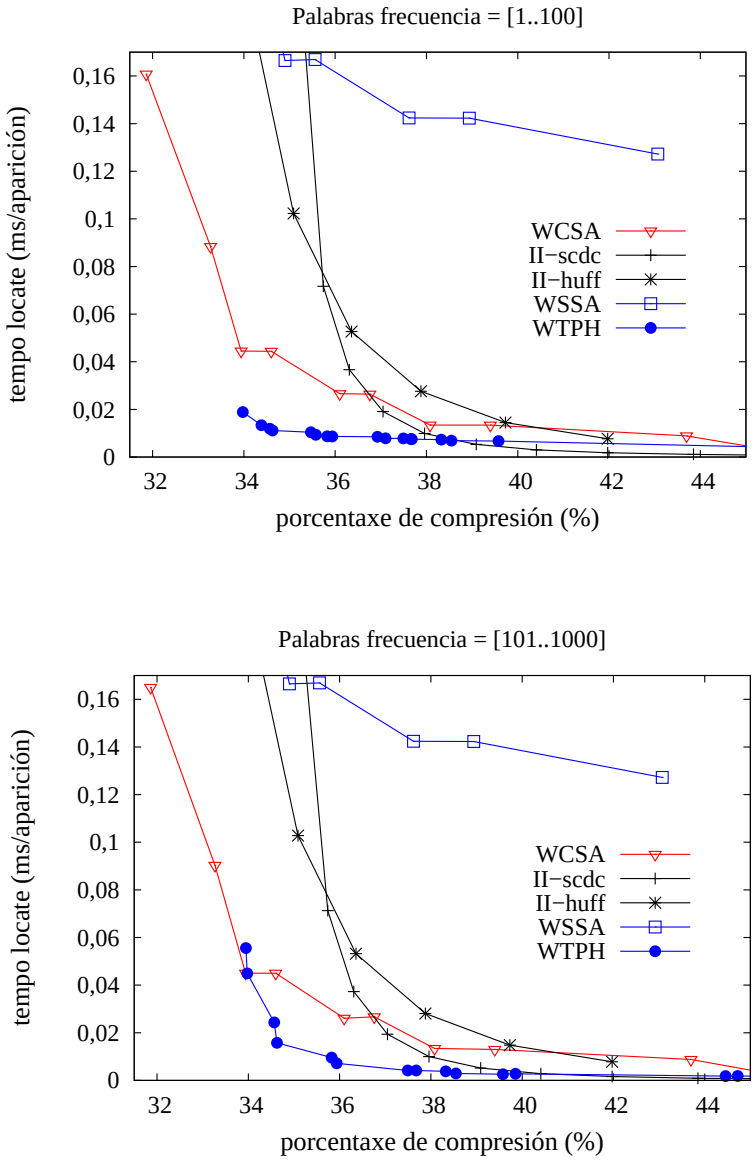

Figura 4.4: Comparativa espazo-temporal para localizar palabras individuais de frecuencia baixa (arriba) e media-baixa (abaixo) para diferentes índices.

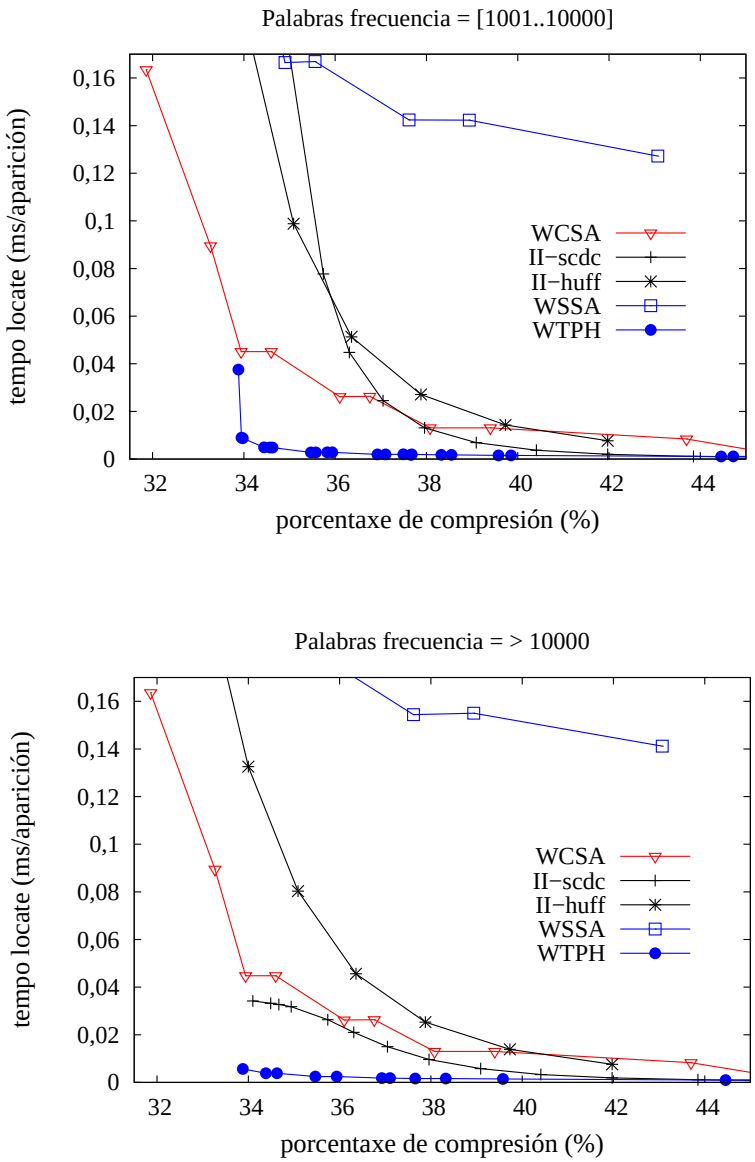

Figura 4.5: Comparativa espazo-temporal para localizar palabras individuais de frecuencia media-alta (arriba) e alta (abaixo) para diferentes índices.

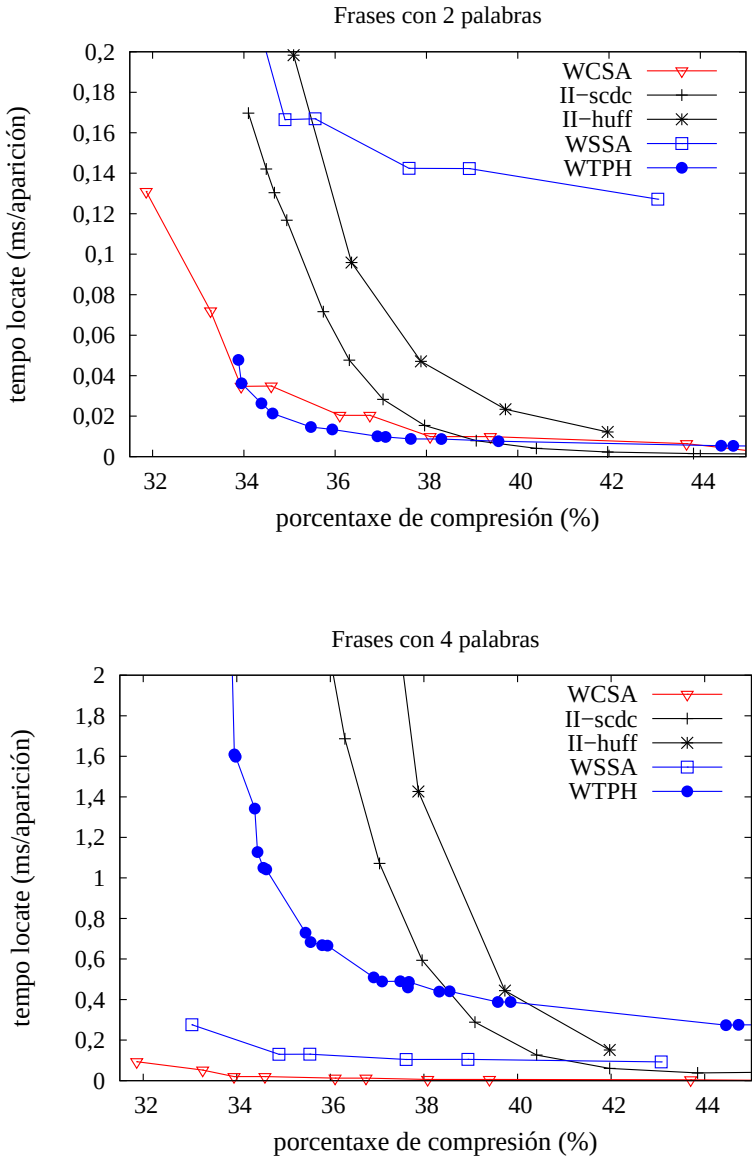

Figura 4.6: Comparativa espazo-temporal para localizar frases de dúas (arriba) e catro (abaixo) palabras para diferentes índices.

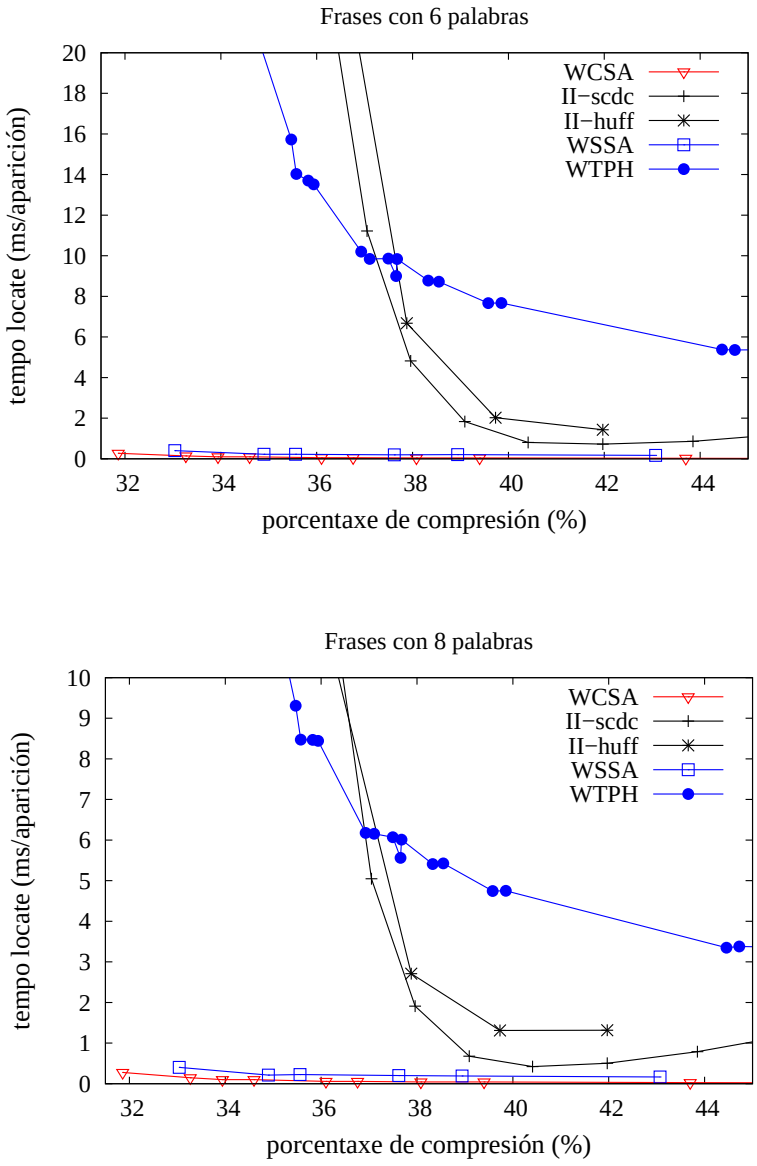

Figura 4.7: Comparativa espazo-temporal para localizar frases de seis (arriba) e oito (abaixo) palabras para diferentes índices.

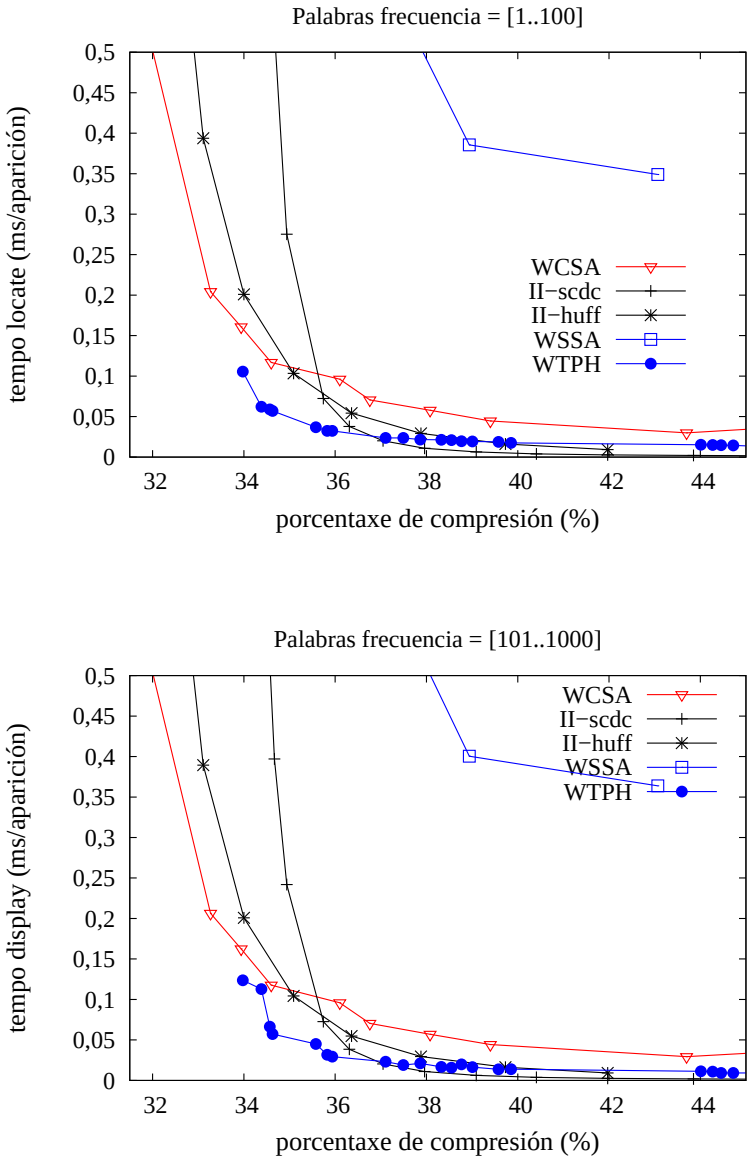

Figura 4.8: Comparativa espazo-temporal para mostrar fragmentos de 20 palabras ao redor de aparicións de palabras individuais de frecuencia baixa (arriba) e media-baixa (abaixo) para diferentes índices.

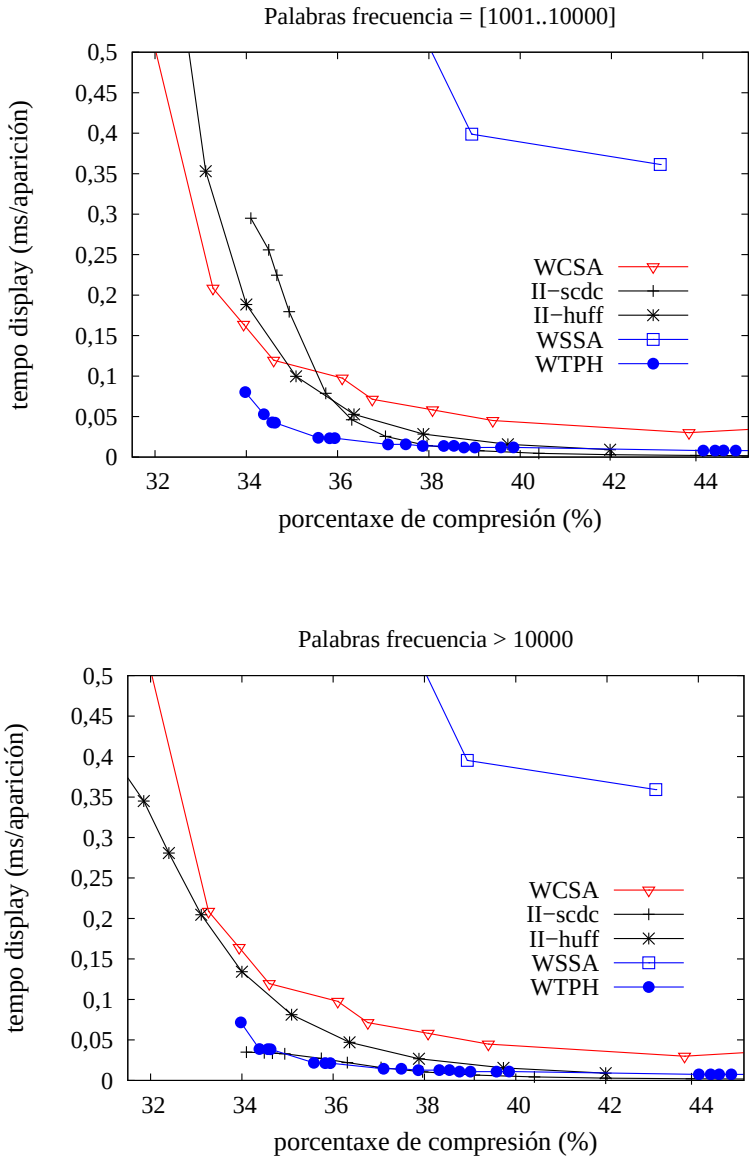

Figura 4.9: Comparativa espazo-temporal para mostrar fragmentos de 20 palabras ao redor de aparicións de palabras individuais de frecuencia media-alta (arriba) e alta (abaixo) para diferentes índices.

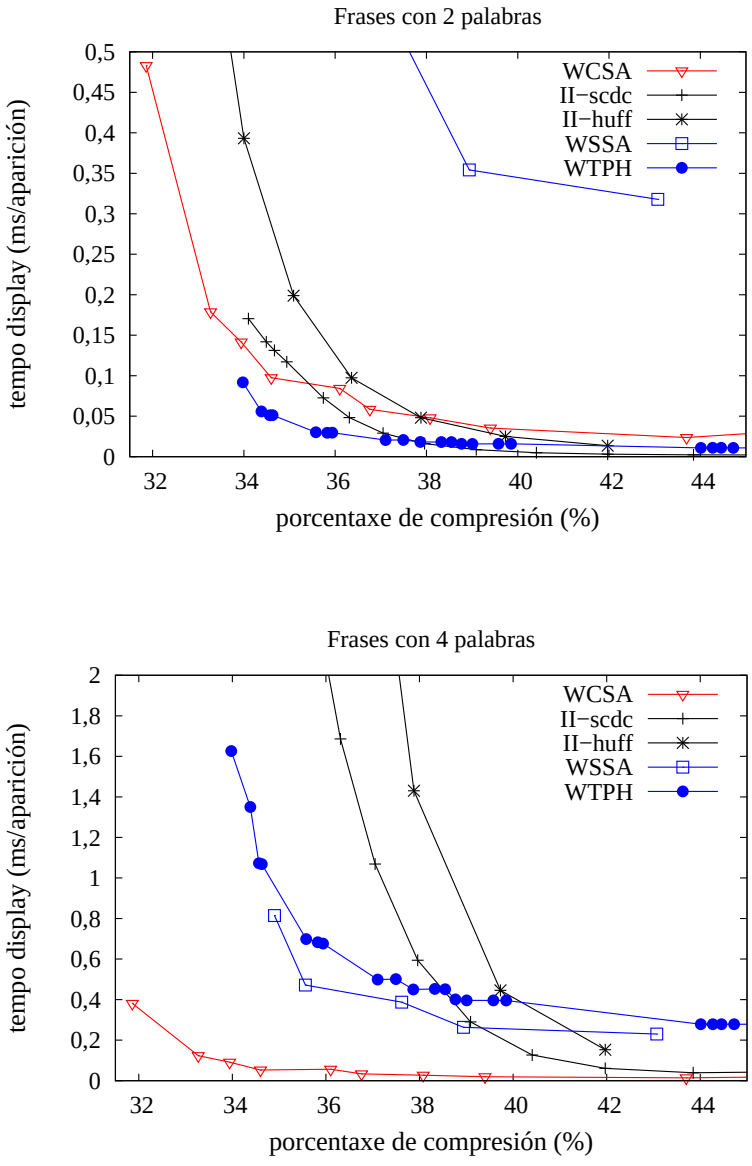

Figura 4.10: Comparativa espazo-temporal para mostrar fragmentos de 20 palabras ao redor de aparicións de frases de dúas (arriba) e catro (abaixo) palabras para diferentes índices.

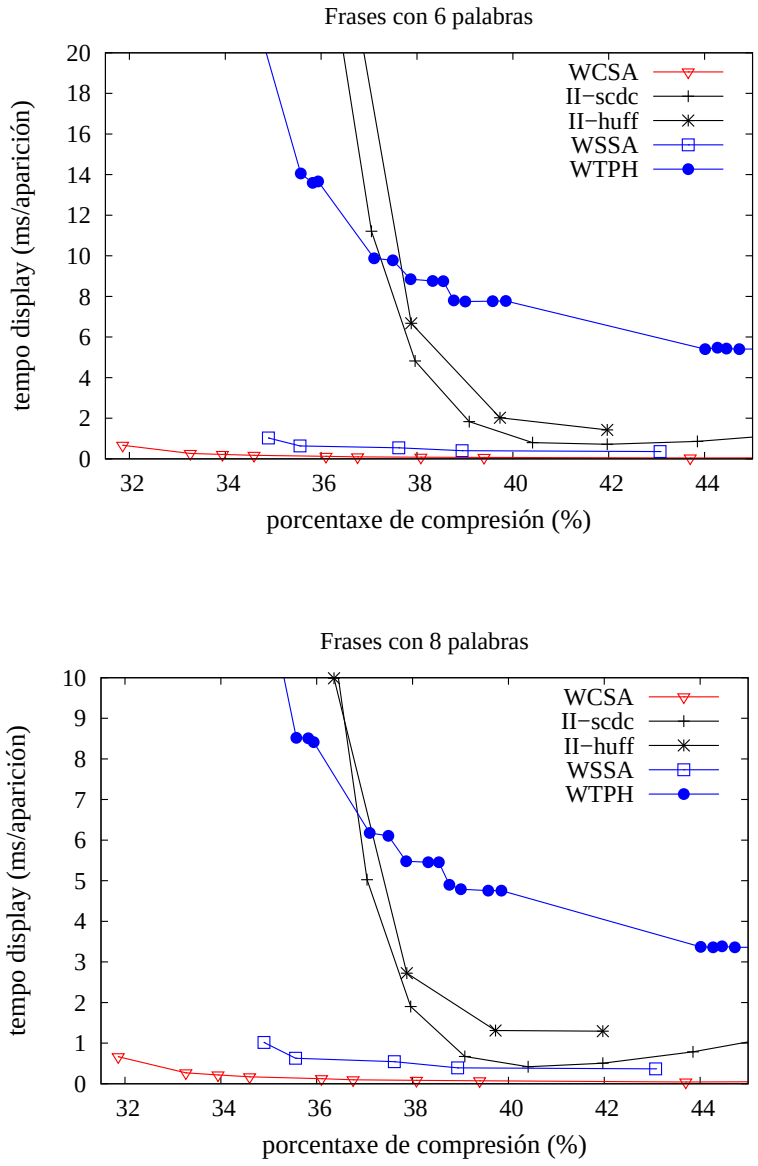

Figura 4.11: Comparativa espazo-temporal para mostrar fragmentos de 20 palabras ao redor de aparicións de frases de seis (arriba) e oito (abaixo) palabras para diferentes índices.

Cadro 4.2: Resultados comparando `FWCSA` con `FII`.

		`FWCSA`$_i$			`FII`$_i$		
		i=1	i=2	i=3	i=1	i=2	i=3
	Ratio (%)	41,42	38,84	37,54	41,32	38,93	37,50
		Locate					
Palabras	1–100	0,030	0,058	0,070	0,042	0,161	0,503
(rango	101–1000	0,030	0,059	0,070	0,019	0,074	0,200
frec)	1001–10000	0,030	0,058	0,069	0,021	0,089	0,171
	10000+	0,028	0,057	0,067	0,011	0,020	0,022
	2	0,027	0,054	0,063	0,044	0,118	0,159
Frases	4	0,030	0,058	0,069	0,026	0,064	0,089
#palabras	6	0,032	0,062	0,074	0,077	0,304	0,485
	8	0,044	0,059	0,074	3,086	15,551	27,795
		Snippet					
Palabras	1–100	0,086	0,148	0,160	0,041	0,161	0,512
(rango	101–1000	0,087	0,151	0,161	0,022	0,078	0,204
frec)	1001–10000	0,085	0,149	0,159	0,024	0,093	0,174
	10000+	0,083	0,145	0,155	0,014	0,023	0,025
	2	0,078	0,139	0,148	0,047	0,121	0,163
Frases	4	0,085	0,148	0,158	0,029	0,067	0,092
#palabras	6	0,092	0,159	0,170	0,080	0,307	0,486
	8	0,084	0,153	0,162	3,110	15,463	27,717

supera ao `WTPH` na busca de frases de lonxitude media ou longa, sendo unha orde de magnitude máis rápido. Ademais o `WCSA` permite baixar as porcentaxes de compresión até taxas que o `WTPH` non chega (o `WTPH` ocupa como mínimo un 33,32 % do texto, obtendo este valor se non se inclúen estruturas que aceleren as buscas e polo tanto obtendo rendementos moi pobres).

Os resultados para a extracción de fragmentos ao redor de aparicións de palabras (*display*), que se inclúen nas figuras 4.8–4.11, son similares aos de localización, aínda que neste caso a vantaxe que obtén o `WCSA` con respecto ao `WTPH` diminúe lixeiramente, xa que a extracción é máis rápida no `WTPH`.

Tamén comparamos o `WCSA` cunha aproximación chamada `TH+AFFM` [FNP08] que consiste nunha compresión orientada a palabras seguida dunha autoindexación orientada a carácter. Comparamos a velocidade de localizar frases compostas por 4 palabras (outros conxuntos de probas dan resultados semellantes) axustando o `WCSA` para que utilice a mesma cantidade de memoria que o `TH+AFFM` con diferentes

combinacións de valores de parámetros para ambos os dous métodos. En todos os casos obtemos que as buscas no `WCSA` son unhas cinco veces máis rápidas.

Capítulo 5

Compresión orientada a frases

Tense demostrado que a compresión orientada a palabras sobre textos en linguaxe natural supón unha boa alternativa no balance entre a porcentaxe de compresión e a velocidade, obtendo taxas de compresión próximas ao 30 % e unha descompresión moi rápida. Ademais este tipo de compresión permite buscas rápidas sobre o texto comprimido usando algoritmos de tipo Boyer-Moore. Estes compresores baséanse en procesar símbolos de lonxitude fixa (palabras) para posteriormente asignarlles unha secuencia de bytes de lonxitude variable, seguindo unha aproximación fixa-a-variable.

A nosa proposta consiste nun novo compresor variable-a-variable (`V2VDC`) que usa palabras e frases como símbolos de entrada, e codifica estes símbolos cun esquema de lonxitude variable. As frases son elixidas usando a información do *Longest Common Prefix (LCP)* do array de sufixos do texto, de maneira que se favorecen as frases longas e frecuentes. Obtemos porcentaxes de compresión próximas ás doutros compresores, como `p7zip` e `ppmdi`, superando ao `bzip2`, e 8–10 puntos porcentuais menos que o compresor orientado a palabras equivalente. Ademais, `V2VDC` é o que descomprime máis rápido e permite buscas directas de forma eficiente no texto comprimido, sendo tamén o máis rápido nalgúns casos.

O capítulo organízase da seguinte maneira. Na sección 5.1 profundamos na motivación desta investigación. Posteriormente na sección 5.2 detallamos as particularidades do noso novo método de compresión, o `V2VDC`. Finalmente na sección 5.3 incluímos a avaliación experimental da proposta presentada.

5.1. Motivación

Tal e como se explicou no capítulo 2, o crecemento das bases de datos textuais ten propiciado un aumento considerable no interese de novas técnicas de compresión de textos capaces de reducir considerablemente o seu tamaño de almacenamento, á vez que manteñen a capacidade de xestionar esas grandes bases de datos textuais en formato comprimido.

Coa aparición dos primeiros dous compresores de texto orientados a palabra [BSTW86, Mof89] demostrouse que os compresores que usan un modelo semiestático de orde cero e orientado a palabra son capaces de reducir as coleccións de texto a aproximadamente un 25 % do seu tamaño orixinal, combinado cun codificador Huffman orientado a bit [Huf52], ou a aproximadamente un 30 %, no caso de utilizar un Huffman orientado a byte [MNZB98, MNZBY00].

A compresión obtida por métodos semiestáticos de orde cero e orientados a byte está acotada inferiormente polas porcentaxes obtidas por *Plain Huffman*, de forma que estas técnicas non poden competir con compresores máis potentes como `p7zip`, `bzip2`, ou aqueles baseados en PPM [CW84]. Coa proposta presentada neste capítulo intentamos superar estas limitacións permitindo que o modelo non só conteña palabras, senón tamén frases, o que nos permitirá representar unha frase enteira cun só código.

O éxito do compresor dependerá fundamentalmente da súa habilidade para escoller boas frases. Este é un problema coñecido no ámbito máis xeral de compresores baseados en gramáticas. Dado que encontrar a gramática máis pequena para un texto é un problema NP-completo [CLL+05] xurdiron diferentes heurísticas, como LZ78 [ZL78], `re-pair` [LM00], ou Sequitur [NMWM94], entre moitas outras [AL00, Ryt03, CLL+05]. Por exemplo, `re-pair` [LM00] xunta pares de frases de forma recursiva realizando múltiples pasadas sobre o texto, formando unha nova frase co par de símbolos máis frecuentes en cada pasada. Outra aproximación [AL00] detecta todas as frases non sobrepostas no texto de entrada e usa unha función de ganancia para medir a bondade desas frases.

Nós utilizaremos unha aproximación similar a esta última [AL00], de forma que as nosas frases son planas (non conteñen a outras). Usamos a información do *LCP* do array de sufixos do texto para seleccionar as frases, baseándonos na súa lonxitude e frecuencia, primando as frases longas e impoñendo un limiar mínimo de frecuencia ou ganancia. Por último, aplicamos un modelo de orde cero e unha codificación `ETDC` á secuencia de frases.

5.2. Variable-to-Variable Dense Code

O noso novo compresor, chamado Variable-to-Variable Dense Code (`V2VDC`), está basicamente composto por un modelo baseado en frases combinado cun codificador `ETDC`. Consideramos como frase calquera secuencia de polo menos dúas palabras que aparecen *minFrec* ou máis veces no texto orixinal. A fase de modelado detecta as frases "boas" coa axuda dun array de sufixos [MM93] e unha estrutura para o LCP que se explica a continuación.

Compresión. O proceso consta dos seguintes pasos:

- *Parseo do texto e selección das frases candidatas*: Nesta fase identificamos todas as frases candidatas no texto fonte T. O resultado desta fase é un vocabulario de palabras e unha lista de frases candidatas.

Empezamos pola identificación das diferentes palabras e as súas frecuencias en T, obtendo un vocabulario de palabras ordenado alfabeticamente (V_p). Usamos V_p para crear unha representación *tokenizada* T_{ids} de T, onde cada palabra é representada por un enteiro (o seu identificador id).

O seguinte paso consiste na creación dun array de sufixos (A) sobre T_{ids}, e dunha estrutura LCP. O LCP almacena, para cada posición $2 \leq j \leq |T_{ids}|$ en A, a lonxitude do prefixo común máis longo entre os sufixos apuntados por $A[j]$ e $A[j-1]$. Un percorrido por LCP permite xuntar todas as frases candidatas de T_{ids}, coa súa lonxitude e número de aparicións. Cada frase maximal[1] de lonxitude ≥ 2 que aparece $\geq$ *minFrec* veces é unha frase candidata. O conxunto de frases candidatas corresponde exactamente aos nodos da árbore de sufixos con polo menos *minFrec* occurrencias, e polo tanto hai $O(|T_{ids}|/minFrec)$. A lista de frases candidatas resultante (LF) ordénase decrecentemente por lonxitude, rompendo os posibles empates ordenando decrecentemente por frecuencia.

- *Recompilación do dicionario de frases final e produción dunha representación baseada en frases (T_f) de T*. Incluímos tanto palabras como frases nun dicionario de frases común *dic*.

Empezamos con $T_f = T_{ids}$. Despois percorremos LF e, para cada frase candidata f_i, comprobamos a súa frecuencia en T contando unicamente as aparicións de f_i que non se superpoñan con frases xa incluídas, que son marcadas cun bitmap de tamaño $|T_{ids}|$. Se f_i aínda merece ser incluída no dicionario final de frases *dic*, marcamos as súas aparicións como "usadas" no bitmap, diminuímos os valores de frecuencia das palabras que contén en V_p e substituímos as aparicións da frase en T_f. Se non

[1] Con "maximal" referímonos a que non se inclúen frases máis curtas que aparecen nas mesmas posicións que frases máis longas

merece ser incluída, probamos prefixos máis curtos de f_i de forma sucesiva até que finalmente a descartamos (o número total de veces pode ser tan alto como o número de nodos do *trie* de sufixos de T, aínda que isto é pouco probable na práctica).

A condición máis sinxela para aceptar ou rechazar f_i depende só de se se cumpre que $frec(f_i) \geq minFrec$. Unha heurística máis sofisticada podería estimar a ganancia na compresión obtida coa inclusión de f_i, que se calcularía da seguinte forma $(bytes_{antes} - bytes_{despois}) * (frec(f_i) - 1) - 2$, onde: *a)* $bytes_{antes} = \sum_j |C_{p_j}|$ é o tamaño dos códigos para as palabras individuais p_j que aparecen en f_i cando f_i é rechazada; *b)* $bytes_{despois} = |C_{f_i}|$ supoñendo que f_i é aceptada; e *c)* -1 e -2 son debidos ao custo de engadir f_i a *dic* (como se ve a continuación). Para permitir a estimación do valor $|C_x|$, *dic* manterase ordenado por frecuencia.

- *Codificación e substitución de códigos.*

Usamos o codificador `ETDC` para proporcionarlle un código a todos os símbolos. Os códigos densos semiestáticos [BFNP07], e en particular `ETDC`, usan un esquema de codificación simple que marca o final de cada código, reservando 1 bit por cada byte do código. Desta maneira, asignan un código de 1 byte aos 128 símbolos máis frecuentes, un código de 2 bytes aos seguintes 128^2 símbolos máis frecuentes, e así sucesivamente. Así, a lonxitude dun código depende só do rango de posicións ao que pertence o símbolo dentro do vocabulario ordenado por frecuencia ($1 \ldots 128$, $128 + 1 \ldots 128 + 128^2$ etc.) e permite codificar e descodificar sobre a marcha, $C_i \leftarrow encode(i)$ e $i \leftarrow decode(C_i)$, en tempo $O(\log(i)) = O(|C_i|)$ [BFNP07].

Polo tanto, ordenamos *dic* por frecuencia para coñecer o número de códigos de diferente lonxitude (1 byte, 2 bytes etc.) que se usarán para codificar as palabras e as frases. Antes de codificar, cada rango correspondente a códigos da mesma lonxitude reorganízase, por razóns prácticas que se explicarán posteriormente, do seguinte xeito: movemos as palabras ao principio e as frases ao final, e ordenamos finalmente as frases segundo a súa primeira aparición en T_f.

A continuación procédese á codificación das palabras e frases. Percórrese T_f e substitúese cada $id = T_f[i]$ polo seu correspondente código para obter o texto comprimido. A única excepción a esta substitución é no caso da primeira aparición dunha frase, xa que neste caso substitúese pola secuencia de códigos das palabras que a compoñen.

Finalmente, inclúese *dic* nunha cabeceira que se almacena xunto co texto comprimido. As palabras almacénanse explicitamente e as frases mediante unha referencia á posición relativa na que aparecen por primeira vez no texto comprimido e a súa lonxitude (en palabras). Esta información é suficiente para as frases, debido a que xa aparecen no texto comprimido. Inclúese tamén o número de palabras e frases en cada rango de lonxitudes de códigos (1 byte, 2 bytes etc.), información que ocupa só uns poucos enteiros.

Para aforrar espazo, comprímese a secuencia de palabras con `p7zip`. As posicións relativas das frases en cada rango, que foron ordenadas antes da fase de codificación, represéntase de forma diferencial utilizando códigos `Rice` [WMB99], e as lonxitudes das frases codifícanse cun Huffman orientado a bits.

Descompresión. Comézase coa recuperación de *dic* e da secuencia de palabras en texto plano, que chamaremos *vocP*. Para cada entrada de *dic* mantemos un par $\langle ptr, lonx \rangle$. Para as palabras, eses valores almacenan a posición dentro de *vocP* e a lonxitude da palabra (en caracteres). No caso das frases, o par contén inicialmente a posición relativa da primeira aparición da frase no texto e o número de palabras que a compoñen. Posteriormente, unha vez que a primeira aparición da frase é descodificada, o par conterá a posición da frase descomprimida no texto e a súa lonxitude (en caracteres).

Mentres se procesa *dic*, vaise construíndo un array auxiliar *offsets*. Para cada frase na posición *id* de *dic*, este array contén unha entrada $\langle id, pos \rangle$ onde *pos* é a posición onde esa frase aparece por primeira vez. Despois o array *offsets* ordénase de forma crecente pola compoñente *p*. Esta ordenación é moi sinxela, xa que as frases codificadas con 1 byte, 2 bytes etc. xa foron almacenadas ordenadas por posición relativa na cabeceira.

Para a descompresión percórrese o texto comprimido, descodificando un código de cada vez usando o array *offsets* para coñecer a posición da primeira aparición das frases en caso necesario. Basicamente o descompresor actúa de forma análoga a `ETDC`, aplicando a descodificación $id \leftarrow decode(C_{id})$ e emitindo o contido de $dic[id].ptr$. A diferenza consiste en que cada vez que a descompresión alcanza unha posición apuntada pola seguinte entrada *j* no array *offsets* ($offsets[j].pos$), feito que acontece cando se detecta a primeira aparición da frase $x{=}offsets[j].id$, o descompresor descodifica os seguintes $dic[id].lonx$ códigos que compoñen esa frase. Finalmente, emítese *x*, increméntase *j* e actualízase o $dic[id]$ como corresponde.

É posible a descompresión aleatoria simplemente comprobando se $id \leftarrow decode(C_{id})$ é unha palabra ou unha frase. Esta comprobación é directa, xa que na cabeceira está a información dos rangos das palabras e frases codificadas con 1 byte, 2 bytes etc. Cando *id* é unha frase pódese descomprimir accedendo a $dic[id].ptr$ e descodificando as seguintes $dic[id].lonx$ palabras seguintes. Neste caso non se modifica o contido de $dic[id]$, xa que o texto segue en formato comprimido.

Buscas. Cando se usa un compresor semiestático orientado a palabras as buscas directas son posibles simplemente comprimindo o patrón e buscándoo no texto comprimido. Nun contexto *variable-a-variable* non é posible buscar só o código da palabra buscada, senón que é necesario buscar os códigos de todas as frases que conteñen esa palabra. Se se busca unha frase debemos buscar tamén os códigos de

todas as frases compatibles que conteñen unha subcadea do patrón buscado, xa que estas frases poden ser combinadas con outras palabras ou frases para conformar o patrón completo desexado.

O noso algoritmo de busca está baseado en *Set-Horspool* [NR02]. Cando buscamos un patrón P formado por unha soa palabra, inicialmente incluímos o seu código C_P na árbore de busca de Horspool. Posteriormente, cada vez que encontramos unha coincidencia, esta devólvese e tamén se comproba se a aparición de C_P aparece dentro da primeira aparición dunha frase f. Neste caso habería que engadir C_f á árbore de busca. Polo tanto, avánzase no texto con Horspool e de forma simultánea no array *offsets*, tal e coma se facía na descompresión.

As buscas de frases resólvense buscando a súa palabra menos frecuente no texto comprimido. Novamente, despois de cada coincidencia débese comprobar se forma parte da primeira aparición dunha nova frase f. Nese caso, se f é compatible co patrón, engádese o seu código C_f á árbore de busca. Para cada patrón incluído na árbore de busca mantemos: *a)* o seu código; *b)* o número de palabras que faltan a súa esquerda e a súa dereita para completar o patrón da frase completo; e *c)* o número de veces que o patrón aparece dentro de f. Para devolver as aparicións débese manter tamén a posición relativa do patrón dentro da frase f.

Cada vez que se encontra unha coincidencia cun código C_i na árbore de busca, engádese o número de veces que contén o patrón P. Ademais compróbanse os códigos que faltan antes e despois de C_i, de forma que se coinciden cos indicados para esa entrada devólvese unha nova aparición do patrón.

5.3. Resultados experimentais

Usamos un corpus textual grande extraído de TREC-2: Ziff Data 1989–1990 (ZIFF), así como dous corpus de tamaño medio de TREC-4, máis concretamente o Congressional Record 1993 (CR) e o Financial Times 1991(FT91). Como colección pequena usamos o corpus Calgary.

Nos experimentos usamos o noso compresor co método do limiar *minFrec* ao que denominamos (`V2VDC`), e tamén unha variante usando unha heurística de ganancia máis complexa, á que chamamos ($\texttt{V2VDC}_H$), e os comparamos con compresores coñecidos do estado da cuestión como `ETDC`[2], gzip[3], p7zip[4], bzip2[5], `re-pair`[6] e

[2] `http://vios.dc.fi.udc.es/codes/`
[3] `http://www.gnu.org/software/gzip/`
[4] `http://www.7-zip.org`
[5] `http://www.bzip.org`
[6] `https://github.com/rwanwork/Re-Pair`, combinado cun codificador Huffman orientado a bits, `http://people.eng.unimelb.edu.au/ammoffat/mr_coder/`

Cadro 5.1: Número de frases con lonxitude de código 1, 2 e 3 bytes para cada valor de *minFrec* e porcentaxe de compresión obtida por cada variante do compresor para o corpus CR.

	V2VDC				V2VDC$_H$			
	Lonx. Código (bytes)				Lonx. Código (bytes)			
	1	2	3		1	2	3	
minFrec	Número de Frases			% Comp.	Número de Frases			% Comp.
2	11	6.452	617.388	25,412 %	1	5.605	421.099	24,601 %
3	18	6.742	370.156	24,074 %	5	6.377	272.405	23,532 %
5	19	7.228	193.662	23,429 %	9	7.052	128.516	22,994 %
6	19	7.290	154.957	23,390 %	10	7.190	99.475	22,965 %
7	19	7.464	128.487	23,431 %	10	7.303	80.546	22,998 %
10	22	7.709	83.637	23,546 %	13	7.559	49.693	23,132 %
50	30	8.788	6.711	24,256 %	22	8.350	3.351	23,954 %

ppmdi[7].

Mostramos as comparativas das porcentaxes de compresión e das velocidades de compresión e descompresión. Ademais mostramos o rendemento das buscas sobre o texto comprimido con `V2VDC` incluíndo experimentos onde se executan buscas sobre texto comprimido e descomprimido, así como utilizando diferentes técnicas de busca.

A máquina utilizada nos experimentos é unha Intel Core2Duo E6420@2.13Ghz, con caché L1 de 32KB+32KB, caché L2 de 4MB e 4GB de DDR2-800 RAM, sobre a que corría Ubuntu 8.04 de 64bits (kernel 2.6.24-24-generic). Compilamos con gcc versión 4.2.4 e coas opcións `-O9 -m32`. Os tempos mídense en tempo de usuario de CPU.

Axuste do parámetro *minFrec*. En primeiro lugar amosamos como afecta o parámetro *minFrec* (número mínimo de aparicións dunha frase para que se considere candidata) na compresión obtida por `V2VDC`. As figuras 5.1–5.2 mostran a porcentaxe de compresión obtida dependendo do valor de *minFrec* para os catro corpus e considerando as dúas versións do compresor, `V2VDC` e V2VDC$_H$. Pódese ver que, en xeral, valores $minFrec \in [5 \dots 10]$ producen unha boa compresión e que a heurística máis complexa obtén mellores resultados.

O cadro 5.1 mostra, para o corpus CR, o número de frases recompiladas durante a fase de modelado para os tres rangos de lonxitude de código (1, 2 e 3 bytes) que utiliza o codificador `ETDC` en `V2VDC` e V2VDC$_H$. Dunha banda obsérvase que cando se

[7] `http://pizzachili.dcc.uchile.cl`, con opcións por defecto

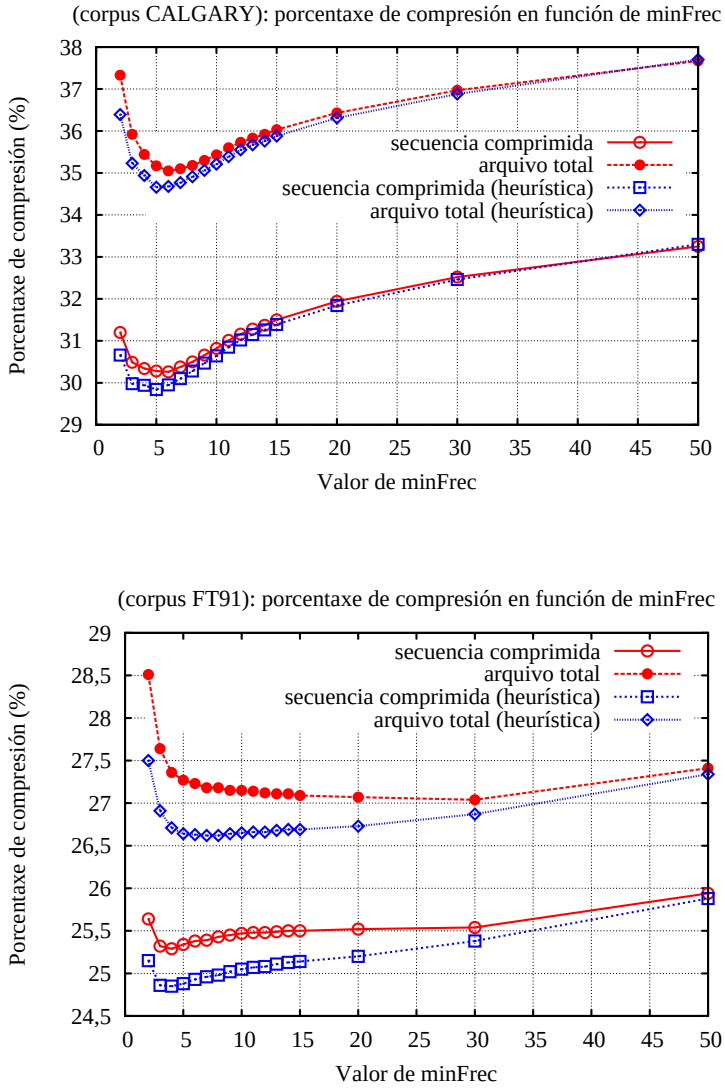

Figura 5.1: Porcentaxe de compresión obtida en función do parámetro *minFrec* nos textos CALGARY e FT91. As curvas denotadas por "*arquivo total*" amosan a compresión total incluíndo o tamaño da cabeceira.

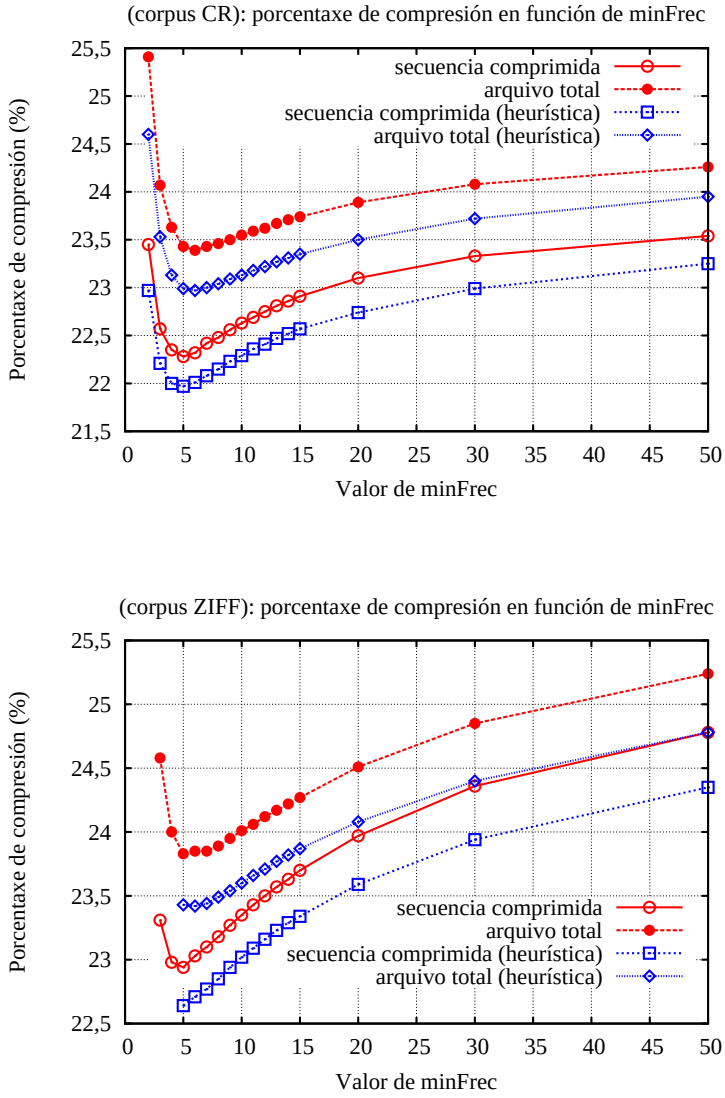

Figura 5.2: Porcentaxe de compresión obtida en función do parámetro *minFrec* CR e ZIFF. As curvas denotadas por "*arquivo total*" amosan a compresión total incluíndo o tamaño da cabeceira.

Cadro 5.2: Comparativa das porcentaxes de compresión para os diferentes métodos.

CORPUS	Size (KB)	ETDC	V2VDC	V2VDC$_H$	re-pair	ppmdi	gzip	p7zip	bzip2
CALGARY	2.081	47,40 %	35,43 %	35,21 %	31,20 %	26,39 %	36,95 %	29,97 %	28,92 %
FT91	14.404	35,53 %	27,15 %	26,65 %	24,00 %	25,30 %	36,42 %	25,53 %	27,06 %
CR	49.888	31,94 %	23,55 %	23,13 %	20,16 %	22,42 %	33,29 %	21,64 %	24,14 %
ZIFF	180.879	33,77 %	24,01 %	23,60 %	20,32 %	23,04 %	33,06 %	22,99 %	25,11 %

utilizan valores altos de *minFrec* se favorece a inclusión de frases que probablemente produzan unha gran ganancia na compresión. Non obstante, o número de frases que aparecen moitas veces non é moi alto, polo que a compresión non se beneficia da ganancia obtida polas frases menos frecuentes. Por exemplo, no corpus CR, só hai unhas 15.500 frases que aparecen polo menos 50 veces, mentres que ao redor de 200.000 frases aparecen máis de cinco veces. Por outra banda pódese ver que como as frases moi longas aparecen poucas veces no texto, o uso dun valor pequeno de *minFrec* permite que sexan escollidas. Con todo, esto provoca tamén a inclusión de frases cortas pouco frecuentes que poden mellorar lixeiramente a compresión (con V2VDC$_H$) ou incluso empeorala se non se usa unha heurística (V2VDC). A heurística só atenúa parcialmente o problema. A pesar de incluír só as frases que producen unha ganancia, (*a*) non controla o problema combinatorio das frases non tan boas, impedindo que boas frases sexan escollidas se se solapan no texto; (*b*) non permite xestionar o problema de que a selección de frases fai decrecer a frecuencia das palabras que a compoñen, de xeito que aplana o histograma de frecuencias e dificulta a compresión de orde cero; e (*c*) só estima a lonxitude final de palabras e frases.

Porcentaxe de compresión. O cadro 5.2 amosa a porcentaxe de compresión obtida polos diferentes compresores probados. Para as variantes do noso compresor configuramos o valor de $minFrec = 10$. Utilizamos este valor xa que a descompresión execútase un 5–10 % máis rápido para $minFrec = 10$ que para $minFrec = 5$. Ambas variantes de V2VDC obteñen boas porcentaxes de compresión cando o tamaño do corpus é suficientemente longo. Os nosos compresores melloran os resultados do ETDC orientado a palabras en 8–10 puntos porcentuais, e os resultados de gzip en máis de 10 (excepto no texto máis pequeno). Pola contra, as variantes de V2VDC son superadas en 4 puntos por re-pair, principalmente debido a que este se beneficia do uso dunha codificación orientada a bit en vez dunha codificación densa. Usando re-pair cun codificador denso a diferenza decrece a 1,5 puntos porcentuais. Os compresores ppmdi e p7zip, que non permiten buscas, superan a V2VDC e V2VDC$_H$ por 1–2 puntos (no texto máis longo), e bzip2 é superado por uns 2 puntos.

Cadro 5.3: Comparativa dos tempos de compresión e descompresión para os diferentes métodos.

CORPUS	ETDC	V2VDC	V2VDC$_H$	re-pair	ppmdi	gzip	p7zip	bzip2
CALGARY	0,128	0,595	0,643	1,910	0,780	0,287	1,610	0,366
FT91	0,652	3,765	6,500	15,554	5,602	1,588	17,932	2,476
CR	2,054	15,425	42,960	69,972	14,441	5,388	65,002	9,550
ZIFF	7,982	76,250	558,970	504,230	55,080	20,667	248,732	34,887

Tempo de compresión (en segundos).

CORPUS	ETDC	V2VDC	V2VDC$_H$	re-pair	ppmdi	gzip	p7zip	bzip2
CALGARY	0,022	0,043	0,049	0,052	0,727	0,034	0,990	0,156
FT91	0,192	0,196	0,203	0,446	4,864	0,197	0,440	0,756
CR	0,584	0,540	0,504	1,516	16,515	0,588	1,354	2,788
ZIFF	2,221	2,324	2,140	5,450	59,058	2,332	5,299	9,717

Tempo de descompresión (en segundos).

Velocidade de compresión e descompresión. O cadro 5.3 mostra os tempos de compresión e descompresión. Durante a compresión V2VDC invirte moito tempo para a construcción das estruturas do array de sufixos e do LCP[8], e V2VDC$_H$ debe abordar a computación da heurística[9] que se usa para seleccionar as boas frases candidatas.

Os compresores máis rápidos superan notablemente as nosas variantes de V2VDC: ETDC é 5–10 veces máis rápido que V2VDC, mentres que gzip and bzip2 son preto de 2–4 e 1–2 veces máis rápidos que V2VDC, respectivamente. V2VDC ten un rendemento similar a ppmdi na maioría dos textos, coa excepción do corpus ZIFF. Se o comparamos cos mellores compresores, en termos de porcentaxe de compresión, vemos que V2VDC é entre 2 e 6 veces máis rápido que re-pair e p7zip (que usa dúas CPUs na nosa máquina de probas tanto na compresión como na descompresión). Polo tanto, podemos resumir que V2VDC$_H$ é normalmente máis rápido que re-pair and p7zip (excepto para o corpus ZIFF) e máis lento que o resto.

Con respecto á descompresión, V2VDC encóntrase entre os descompresores máis rápidos. Benefíciase dos seus mellores resultados en porcentaxe de compresión e da rapidez do seu algoritmo de descompresión (similar ao de ETDC), obtendo resultados

[8] Usamos *qsort* para construír o array de sufixos e unha aproximación simple de forza bruta para o LCP. Poderíanse usar algoritmos máis sofisticados [MF04, KLA+01] para acelerar estas tarefas.

[9] Debemos manter o vocabulario ordenado por frecuencia para calcular o tamaño do código para cada frase ou palabra. A velocidade de compresión pode acelerarse utilizando ideas de traballos relacionados [BFNP10].

comparables aos obtidos polos descompresores coñecidos máis rápidos, como son ETDC e gzip. A única excepción dase no corpus pequeno CALGARY, xa que V2VDC dedica a maior parte do tempo de descompresión a recuperar a cabeceira. A outra variante, $\texttt{V2VDC}_H$, obtén incluso mellores resultados debido á súa mellor porcentaxe de compresión e polo feito de que manexa un número menor de frases.

Tempo de busca. Buscamos patróns de palabras individuais elixidas aleatoriamente do vocabulario do corpus ZIFF, seguindo o modelo [MNZBY00] onde cada palabra se busca con probabilidade uniforme. Estes patróns clasifícanse en tres rangos de frecuencia: baixa, media e alta. O cadro 5.4 mostra as lonxitudes medias dos patróns de cada rango e as medicións de tempos. Consideramos dous escenarios: por unha banda consideramos buscas realizadas sobre o texto plano usando a nosa propia implementación do algoritmo de *Horspool* [NR02]. Pola outra banda, realizamos buscas no texto comprimido con ETDC utilizando unha adaptación do algoritmo de Horspool[10]. Comparamos estas buscas coas obtidas sobre o texto comprimido con V2VDC e $\texttt{V2VDC}_H$.

No caso das buscas sobre texto comprimido non estamos incluíndo na medición o tempo necesario para cargar a cabeceira do fichero comprimido, senón só o procesamento do mesmo. Isto é debido a que esta carga só se fai unha vez e amortízase cando se realizan múltiples buscas. O tempo de carga é 280, 232 e 90 milisegundos respectivamente para V2VDC, $\texttt{V2VDC}_H$ e ETDC.

Como se demostra en traballos previos [BFNP07], as buscas sobre texto comprimido son máis rápidas que as buscas sobre texto plano. A única excepción é que *Horspool* sobre texto plano supera a V2VDC nas buscas de patróns moi frecuentes. Neste escenario as variantes de V2VDC deben realizar en paralelo buscas de moitas frases, todas as que poidan conter á palabra buscada. Para palabras de frecuencia baixa e media, as variantes de V2VDC son capaces de mellorar os resultados non só do texto plano senón tamén de ETDC. Este resultado ten especial interese xa que ETDC é coñecido por ser a técnica orientada a palabras máis rápida no caso das buscas en texto comprimido [BFNP07].

[10] http://vios.dc.fi.udc.es/codes/

Cadro 5.4: Comparativa do tempo de busca (en milisegundos) para o corpus ZIFF.

info patróns		texto plano	texto comprimido		
Rango Frec	Lonx. media	*Horspool*	ETDC	V2VDC	V2VDC$_H$
1–10	8,06 bytes	155,330	98,246	87,897	88,874
10–10^3	7,82 bytes	209,493	122,848	118,997	97,103
10^3–10^5	7,61 bytes	174,251	155,038	235,858	214,647

Parte II

Contidos dixitais: aplicacións prácticas

Capítulo 6

Panorámica dos contidos dixitais

Podemos definir os contidos dixitais como calquera tipo de información que existe en forma de datos dixitais, e que por tanto pode ser de distinta natureza, como texto, vídeo, sons, imaxes, animacións etc. No ámbito desta tese imos referirnos aos contidos dixitais, de forma máis concreta, como aqueles destinados principalmente ao consumo (lectura, visualización, interacción etc.) por parte das persoas. Neste sentido son contidos dixitais, por exemplo, os documentos e arquivos multimedia que almacenamos nos nosos ordenadores, *smartphones* ou *tablets*; as distintas informacións publicadas en páxinas web e redes sociais; as fotografías e vídeos que tomamos coas cámaras dixitais; os espazos televisivos que xa se transmiten exclusivamente a través de canles dixitais; a música, as películas ou os videoxogos que se distribúen en soporte CD, DVD ou Blu-ray; e así un longo etcétera. É fácil decatarse, pois, que somos grandes consumidores e tamén produtores de contidos dixitais, e que o seu volume medra día a día a un ritmo imparable.

A orixe dos contidos dixitais hai que buscala na coñecida como Revolución Dixital, iniciada a mediados do século XX coa aparición do transistor, e consolidada a finais do século XX e principios do XXI co auxe das ciencias da computación e as tecnoloxías da información e as comunicacións. En todo este tempo vénse producindo unha progresiva transformación marcada polo paso do analóxico ao dixital, que hoxe afecta practicamente a todos os campos e actividades da sociedade. Se ben a información que os seres humanos somos capaces de xerar e percibir a través dos sentidos é exclusivamente analóxica, os ordenadores e sistemas de computación dos que nos servimos para almacenala, compartila e consultala traballan coa información en formato dixital.

Son moitas as vantaxes que ofrece o emprego de tecnoloxías dixitais para o almacenamento, procesamento, transmisión e consumo de contidos, entre as que cabe destacar as seguintes:

- Almacenamento. As tecnoloxías de almacenamento dixital actuais permiten gardar grandes volumes de datos en moi pouco espazo físico, desde logo moi inferior ao necesario para almacenar cantidades de información equivalentes en soportes físicos.
- Replicación. Os contidos dixitais son moi fáciles de duplicar, coa singularidade ademais de que a copia do contido que se obtén é unha réplica exacta, e que polo tanto conserva a calidade do orixinal.
- Transmisión. Os datos en formato dixital ofrecen grandes posibilidades de seren transmitidos a través de redes de comunicación como Internet, polo que xa non resulta significativa a distancia física que poida existir entre produtor e consumidor dos contidos dixitais.
- Accesibilidade. Moitos dos dispositivos de uso común na actualidade, como *smartphones*, *tablets* ou ordenadores, ofrecen unha grande capacidade de acceso ao consumo de contidos dixitais de distinta natureza.
- Durabilidade. Os contidos dixitais son resistentes á degradación polo paso do tempo. Na práctica, a súa persistencia vai estar supeditada á do medio de almacenamento empregado, mais dada a súa capacidade de replicación, resulta sinxelo manter copias de seguridade (*backups*) de respaldo ante posibles continxencias.
- Dispoñibilidade. En parte como consecuencia de todo o anterior, resulta factible obter unha alta dispoñibilidade dos contidos dixitais para o seu acceso tanto local como a través de Internet.

Todas estas vantaxes que ofrecen os contidos dixitais, e que benefician tanto a produtores como aos consumidores, son a causa do gran crecemento que nos últimos anos está a vivir o negocio dos contidos dixitais orientados a consumo.

Un dos sectores que rexistrou un maior crecemento no que a volume de negocio dixital se refire foi, en xeral, o cultural e de lecer e, en particular, o editorial. A evolución das tecnoloxías de acceso á lectura de *ebooks*, tanto mediante dispositivos específicos (*eReaders*) como de uso xeral (*smartphones* e *tablets*), está a mudar as preferencias dos usuarios para o consumo de contidos editoriais. Os dispositivos actuais ofrecen unha boa comodidade para a lectura, mesmo durante períodos continuados, unha gran capacidade para almacenar un catálogo amplo de *ebooks*, e tamén unha boa conectividade que permite aos usuarios acceder a contidos *online*

baixo demanda. Como resultado, o mercado dos contidos editoriais está a sufrir un desprazamento importante desde o libro en papel cara os libros en formato dixital.

Por todo isto, as empresas do sector editorial tradicional víronse na obriga de afrontar unha serie de retos para se adaptar á nova era dixital e garantir a súa supervivencia. O máis importante probablemente sexa a obtención das competencias e tecnoloxías que lles permitan ofrecer unha resposta axeitada e de calidade á crecente demanda de contidos dixitais por parte dos consumidores. Mais non é un paso sinxelo, e unha situación frecuente na que se atopan as editoras é a de tratar de manter a publicación en papel como base do negocio e afrontar ocasionalmente algunha aventura dixital cando a situación é propicia ou xorde unha oportunidade de financiamento. No ámbito galego, ben coñecido polo marco en que se desenvolveu este traballo de tese, son poucas as editoras que apostan por levar a cabo unha adaptación decidida do seu negocio cara un modelo produtivo e de distribución dixital.

Os dous aspectos máis problemáticos cos que se atopan as editoras no proceso de adaptación ao dixital son a creación e a distribución de contidos dixitais. Por un lado, a entrada no mundo dos contidos dixitais afecta de forma crítica ao modelo produtivo das empresas do sector. Primeiramente, vanse atopar coa necesidade de dispor e aprender o manexo de novas ferramentas (ou ben as novas funcionalidades que van incorporando as ferramentas tradicionais) para a edición dixital dos contidos. En todo caso, seguramente o máis difícil para o persoal destas empresas non é aprender o manexo dunha nova ferramenta de edición, senón ser capaces de seguir o vertixinoso ritmo de transformacións das tecnoloxías de publicación e consumo de contidos dixitais (formatos, dispositivos etc.), que introduciron seguramente máis cambios no sector editorial en dez anos que os producidos nos cinco séculos anteriores, desde a aparición da imprenta. Por exemplo, unha das principais implicacións que tivo o cambio do soporte en papel ao dixital foi a aparición dos formatos fluídos como EPUB ou MOBI, nos que os contidos das publicacións se adaptan ao tamaño de pantalla dos dispositivos de lectura (*reflowable content*), o que obriga a repensar o deseño das publicacións. Por exemplo, foi preciso redefinir por completo a tarefa de maquetación tradicional. Se até o momento, maquetar unha publicación consistía en definir a organización dos distintos elementos de contido dentro do espazo físico das páxinas, agora consiste en organizar os elementos de contido para que resulten accesibles desde calquera dispositivo de lectura, independentemente do seu tamaño de pantalla[1]. Por desgraza, os formatos fluídos que funcionan moi ben para *ebooks* literarios formados basicamente por texto, non ofrecen resultados satisfactorios en outro tipo de publicacións como revistas, cómics ou libros infantís ilustrados, nas que os contidos gráficos teñen unha gran relevancia. Como consecuencia disto, o principal

[1] Nos formatos fluídos, a páxina é un concepto artificial que non depende da publicación, senón do dispositivo de lectura, e que representa o fragmento de contido que entra na pantalla nun momento dado.

estándar aberto de publicación dixital, EPUB, define na súa última versión unha serie de propiedades para dar soporte a publicacións que precisan unha maquetación fixa (*fixed-layout*[2]), no que se pode ver como unha volta atrás cara unha maquetación en tamaño fixo, relativo a píxeles de pantalla en lugar de centímetros de papel, pero fixo ao fin e ao cabo.

Isto que só é un exemplo, dá unha idea das dificultades que para as empresas do sector editorial supón manterse ao día dos últimos avances tecnolóxicos e das novas tendencias en publicación dixital, e da problemática que xorde ao redor da edición dos seus contidos en dixital. Ante esta situación, as editoras atopáronse coa necesidade de acudir a empresas de desenvolvemento software para que estas levasen a cabo a transformación dos seus contidos a formatos estándar de publicación dixital, dando lugar na maioría dos casos a proxectos editoriais moi custosos e non rendibles en termos económicos. A todo o xa comentado hai que sumar tamén a entrada no negocio de grandes compañías tecnolóxicas que promoven un mercado dos contidos dixitais altamente competitivo e incrementan o nivel de esixencia dos consumidores en canto á calidade dos mesmos. Isto obriga tamén a unha maior especialización e á formación continuada do persoal encargado da creación dixital dos contidos editoriais.

Con respecto á distribución, e seguindo a tendencia iniciada por outros sectores da industria cultural e do lecer como a música ou o cine, nos que a comercialización de contidos dixitais en soporte físico é cada vez menor, as solucións pasan a día de hoxe por ofrecer alternativas de consumo a través de Internet. Ao igual que ocorría co modelo de produción, a distribución de contidos dixitais por Internet cambia radicalmente o modelo de distribución tradicional a través de empresas loxísticas que se encargan do transporte dos libros en papel a librerías e outros puntos de venda. As editoras precisan agora plataformas de *eCommerce* nas que publicar os seus catálogos de *ebooks*, e desde as que distribuílos entre os usuarios compradores a través da Rede.

Algunhas editoras fixeron unha primeira aproximación á distribución dos seus libros en formato dixital (normalmente o PDF de impresión) por Internet, integrando métodos de pago electrónico como PayPal nas súas propias páxinas web, o que lles permitía ofrecer os seus libros directamente aos clientes eliminando intermediarios, mais a gran maioría de editoras non estaba disposta a permitir a descarga dos seus libros nun formato que resultaba sinxelo de replicar e distribuír indiscriminadamente. Xorde pois a necesidade de dispor de solucións tecnolóxicas que permitan ás editoras a distribución controlada dos seus *ebooks* e que garantan que só os usuarios autorizados teñan acceso á lectura, e a resposta son os sistemas DRM (*Digital Rights Management*). Na práctica, a alternativa que existe para implantar unha solución DRM para a distribución de *ebooks* desde a web dunha editora é a tecnoloxía de Adobe (Adobe Digital Editions Protection Technology, ADEPT), mais esta ten

[2] `http://www.idpf.org/epub/fxl/`

un custo económico non asumible para a gran maioría de editoras e dificilmente amortizable para calquera. Así, pronto comezan a xurdir plataformas multieditoriais que actúan de intermediarias para a venda e distribución protexida (co DRM de Adobe) de *ebooks*, a cambio dunha comisión que para a editora vai supor renunciar a unha porcentaxe moi importante dos ingresos por vendas. Esta parte será a suma da marxe comercial que fixa a propia plataforma intermediaria polos seus servizos, unido ao custo da protección DRM que irá parar a Adobe. Non dispostas a renunciar a esa parte dos ingresos en favor de Adobe, outras grandes compañías como Google, Amazon ou Apple desenvolveron as súas propias plataformas e dispositivos que ofrecen outra alternativa para a venda protexida de contidos dixitais a cambio tamén dunha comisión, polo xeral do 30 %.

Ben sexa a través da integración da solución DRM de Adobe ou ben distribuíndo os contidos a través de plataformas de terceiros, o certo é que unha parte importante das vendas vai caer fóra das arcas da editora. Ademais, os modelos de negocio que unha editora vai poder empregar na venda dos seus *ebooks* son os que proporcionan estas solucións de terceiros, non sendo posible implantar un modelo propio que se axuste mellor ás súas necesidades. Existe unha terceira alternativa, que é o desenvolvemento dunha plataforma propia da editora, que integre un sistema DRM e un modelo de negocio tamén propios, e que polo tanto permita comercializar os *ebooks* sen ter que pagar as elevadas comisións a terceiros que fan de intermediarios. Desenvolver unha plataforma a medida é moi custoso e precisa dunha inversión inicial moi forte, a diferenza do uso dunha plataforma de terceiros onde o custo é proporcional aos ingresos, polo que practicamente ningunha editora afronta un proxecto deste tipo a modo individual[3].

Toda esta problemática ao redor da creación e distribución de contidos editoriais en formato dixital levounos a abrir en Enxenio unha liña específica desde a que levar a cabo proxectos editoriais concretos, mais fundamentalmente, desenvolver produtos e solucións tecnolóxicas avanzadas que nos permitan poñer ao alcance das empresas do sector editorial as ferramentas que lles permitan ser autónomas para a creación de publicacións dixitais e non depender de terceiros para a súa comercialización.

Nos seguintes capítulos preséntanse algunhas das solucións desenvolvidas en resposta ás necesidades do sector á creación (capítulo 8) e distribución (capítulo 7). Para poñer en contexto estas dúas tarefas de creación e distribución de contidos dixitais, na figura 6.1 amósase o ciclo de vida típico dos contidos dixitais distribuídos a través de plataformas *online*.

Partindo dunha idea ou dunha publicación existente en papel, xérase un contido nun formato de publicación dixital (creación). Posteriormente súbese este contido a unha plataforma de distribución web para poñelo dispoñible aos usuarios

[3]Como veremos no capítulo seguinte (7) co caso da Asociación Galega de Editores, si xurdiron algúns proxectos para o desenvolvemento de plataformas multieditoriais deste tipo.

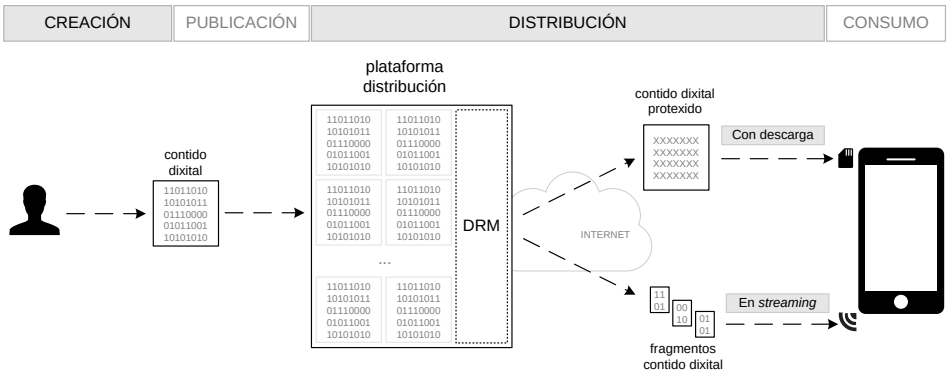

Figura 6.1: Ciclo de vida dos contidos dixitais distribuídos a través de Internet.

(publicación). Cando un usuario accede á plataforma e merca o contido vai poder descargalo (distribución), ben completo (distribución mediante descarga) ou baixo demanda (distribución en *streaming*), para finalmente visualizalo desde algún dos seus dispositivos (consumo).

Capítulo 7

Distribución de contidos dixitais

A distribución de contidos dixitais vén amosando un incremento notable nos últimos tempos, impulsada pola cada vez maior facilidade de acceso da sociedade a Internet e aos dispositivos de consumo dixital. Unha parte importante destes contidos son os pertencentes á coñecida como industria cultural e de lecer, e en particular, ao sector editorial, que atopa en Internet unha canle ideal para a comercialización dos seus libros en formato dixital. Algunhas das vantaxes que Internet ofrece para a comercialización de contidos dixitais son: a redución dos custos de distribución, que permite tamén ofrecer prezos máis reducidos aos clientes; a rapidez na entrega, xa que o usuario pode pagar a través dun medio electrónico seguro e dispor do contido ao momento; a dispoñibilidade ilimitada dos produtos, pola capacidade de replicación dos mesmos debida á súa natureza dixital; e outras como a posibilidade de dispor dunha maior oferta de contidos, as facilidades de busca, o acceso a información completa do produto, a redución do espazo físico necesario para almacenar os contidos, un menor impacto medioambiental (co aforro en papel e no transporte) etc.

Existen multitude de alternativas para a distribución de contidos dixitais, se ben o medio máis habitual na actualidade son as coñecidas como plataformas de distribución, que podemos definir de forma xeral como sistemas software que almacenan un catálogo de contidos dixitais nun ou varios formatos de distribución, e ofrecen aos usuarios os mecanismos de acceso para o consumo de ditos contidos a través de Internet.

Para enfocar a problemática existente na distribución de contidos dixitais a través de Internet foi preciso realizar un estudo en profundidade de diferentes aspectos

como as características dos contidos dixitais (tipoloxía e formatos de publicación), as alternativas tecnolóxicas existentes na actualidade para a súa distribución, os modelos de negocio máis útiles para as editoras ou os hábitos de consumo dos usuarios, resultando os aspectos máis relevantes os que se amosan a seguir:

- Tipo de contidos. Existen plataformas de distribución para contidos dixitais de diferente natureza, como vídeo, música, *ebooks*, videoxogos etc. Na actualidade, é unha tendencia das grandes compañías tecnolóxicas que se dedican á comercialización de contidos dixitais ofrecer plataformas que integran a distribución de diferentes tipos de contidos. No caso dos *ebooks*, como é lóxico, unha parte moi importante do seu contido é texto en linguaxe natural.

- Formato de distribución. Para cada tipo de contido existen, á súa vez, diferentes formatos dixitais de distribución, que condicionan os aplicativos e dispositivos nos que se poden consumir. Por exemplo, no caso dos libros electrónicos algúns dos formatos máis habituais son o EPUB, PDF ou MOBI.

- Tecnoloxía aberta ou propietaria. As plataformas de distribución de contidos dixitais poden estar orientadas á distribución de contidos en formatos estándar, e que polo tanto poden ser consumidos en calquera dispositivo compatible, ou á distribución de contidos en formatos pechados que só poden ser visualizados desde uns dispositivos e aplicacións concretas, e que comercializan os mesmos propietarios da plataforma.

- Modelo de distribución. Basicamente hai dous modelos que son, a descarga, na que o usuario obtén unha copia dixital do contido que poderá consumir en calquera momento nun dispositivo compatible; e mediante *streaming*, na que o usuario non descarga unha copia do contido, senón que o consume directamente a través de Internet, precisando por tanto dispor de conexión durante todo o proceso.

- Distribución por terceiros ou propia. Algunhas plataformas permiten que calquera usuario, previa aceptación das correspondentes condicións de uso da mesma, poida distribuír os seus propios contidos dixitais desde elas (Kindle Store de Amazon, iBooks Store de Apple, Google Play etc.), mentres que outras son desenvolvidas por organizacións particulares para a distribución dos seus propios contidos.

- Modelo de negocio: os mecanismos para monetizar a distribución de contidos dixitais son, ao igual que ocorre co resto de servizos ofertados a través de Internet, moitos e moi variados. Podemos facer unha primeira clasificación das plataformas de distribución de contidos dixitais en dous grandes grupos: gratuítas e de pago. Por gratuítas referímonos a aquelas nas que o usuario non ten que pagar, polo menos de forma explícita, para o consumo dos contidos.

Estas plataformas susténtanse xeralmente en modelos de publicidade, pero tamén poden ser resultado de proxectos institucionais mantidos con fondos públicos, mediante *crowdfunding*, sufragados a través de doazóns, ou unha combinación de varios. En canto ás de pago, que serían aquelas nas que o usuario ten que pagar para o consumo dos contidos, existen multitude de modelos de explotación comercial, como o pago por contido individual, por subscrición, aluguer etc.

Ademais de todas estas cuestións, do estudo anterior determinamos que hai tres aspectos fundamentais que resultan transversais á distribución de contidos dixitais, e que son: a compresión dos contidos (para reducir o seu espazo), a indexación (para permitir buscar información en eles) e a súa protección (para garantir a súa comercialización).

Compresión

É imprescindible para o seu éxito que as plataformas de distribución de contidos dixitais, ao igual que calquera outro sistema que ofreza servizos a través da web, proporcione unha boa experiencia de usuario aos seus clientes. Existen distintas guías para o deseño de interfaces de usuario intuitivas e amigables, e outras recomendacións de usabilidade para o deseño da interacción que deben ofrecer as aplicacións. Mais, probablemente, un dos aspectos que máis inflúen na experiencia de usuario é o tempo de resposta do sistema. Jakob Nielsen, considerado un dos *gurús* da usabilidade web, establece no seu libro *Usability Engineering* [Nie93] tres límites ao tempo de resposta dun sistema, baseados en estudos previos sobre as capacidades perceptivas dos humanos [Mil68] [CRM91], que deben ser tidos en conta na optimización do rendemento de calquera aplicación:

- 0,1 segundos é o límite para que un usuario teña a percepción de que o sistema responde de forma inmediata, e polo tanto non é preciso que envíe ao usuario ningún *feedback* especial mais alá do propio resultado da operación que está a realizar.
- 1,0 segundo é o límite para que o fluxo de pensamento do usuario non se interrompa, aínda que si sexa consciente da demora na resposta. Normalmente non será preciso que o sistema envíe ao usuario ningún *feedback* especial con tempos de resposta entre 0,1 e 1,0 segundos, pero si que vai ocorrer que o usuario perderá a sensación de estar operando directamente sobre os datos.
- 10 segundos é o límite para manter a atención do usuario centrada no diálogo. Para demoras maiores, os usuarios van querer realizar outras tarefas mentres esperan a que a aplicación remate o seu traballo, polo que deberían recibir *feedback* indicando o tempo estimado de finalización.

A conclusión que se extrae de aquí é que a marxe para que o tempo de resposta dun sistema non sexa percibido de forma molesta polos usuarios é moi baixa, e que polo tanto a optimización deste tempo debe ser un aspecto básico a ter en conta no seu desenvolvemento. No caso das plataformas de distribución de contidos dixitais, un factor crítico vai ser o tamaño dos arquivos distribuídos, que terá unha incidencia directa no tempo de descarga (completa ou a en *streaming*) e visualización dos contidos, afectando polo tanto á experiencia do usuario. Por este motivo, resulta moi conveniente o emprego de técnicas de compresión que permitan reducir o tamaño dos arquivos a distribuír.

Indexación

Outra característica desexable dunha plataforma de distribución de *ebooks*, sobre todo cando xestiona un catálogo moi grande, é a de ofrecer distintas funcionalidades de busca tanto sobre os atributos descritivos (título, autoría etc.) como no propio texto dos libros. Para isto, vai ser preciso empregar índices que permitan resolver as buscas de forma eficiente.

En xeral as plataformas de distribución non se preocupan da compresión dos contidos, aínda que ás veces son os propios formatos os que inclúen na súa especificación algún tipo de compresión (por exemplo, o formato EPUB[1] utiliza a compresión con Deflate [Deu96]). En canto a funcionalidades de busca, as plataformas acostuman incluír formularios máis ou menos sinxelos que permiten buscar un *ebook* a través dos seus atributos descritivos (*metadatos*), pero ningunha, ou practicamente ningunha, ofrece a posibilidade de buscar no texto completo dos libros.

Protección

Outro aspecto a considerar nas plataformas que distribúen contidos dixitais de pago, é a integración de solucións para a protección dos dereitos dixitais de ditos contidos (DRM, *Digital Rights Management*), a fin de garantir que só os usuarios autorizados poden acceder a eles. A diferenza dos produtos físicos, os contidos dixitais son facilmente replicables, obtendo unha copia exacta do contido orixinal, e poden distribuírse tamén de forma sinxela e masiva a través de medios como as redes P2P (*peer-to-peer*). Por este motivo, o control de acceso aos contidos dixitais é un aspecto básico das plataformas de distribución comerciais que ofrecen contidos suxeitos a dereitos de autoría ou de explotación por parte das editoras.

Existen diferentes solucións para a distribución protexida dos contidos a través de plataformas dixitais que se caracterizan por ofrecer un maior ou menor nivel de seguridade. Así, podemos establecer unha clasificación dos sistemas DRM en fortes

[1] `http://www.idpf.org/epub/30/spec/epub30-ocf.html`

e lixeiros. As plataformas de distribución que actúan de intermediarias, como as de Amazon ou Apple, incorporan os seus propios mecanismos de protección (xeralmente un DRM forte) como parte integral do servizo que ofrecen ás editoras. No caso de ter que integrar unha solución DRM nunha plataforma de distribución propia, a elección dunha ou doutra modalidade vai atender sobre todo a criterios económicos (unha solución de DRM forte é sensiblemente máis custosa), do crítica que resulte a protección dos contidos nese contexto, ou da facilidade que se quere ofrecer para o seu consumo por parte dos clientes. E é que unha das principais críticas aos sistemas DRM fortes é que non só introducen dificultades para o acceso aos contidos por parte dos usuarios non autorizados, senón que moitas veces dificultan tamén o acceso a aqueles que si o están, obrigándoos a instalar aplicacións de lectura concretas, rexistrarse en servizos de terceiros, limitando o número de dispositivos desde os que poden acceder etc.

En xeral, desde o punto de vista funcional, un sistema DRM inclúe un compoñente que actúa do lado do servidor, e que se encarga de almacenar e controlar os permisos asociados a un *ebook*, e un compoñente na parte cliente, que se integra nas aplicacións de lectura para facer o control dos permisos no momento da lectura, permitindo ou restrinxindo o acceso do usuario ao libro en base aos permisos de que dispón. Para a implantación dunha solución de protección forte dos contidos nunha plataforma de distribución propia, existen alternativas ao desenvolvemento dun sistema DRM específico baseadas no emprego dunha tecnoloxía de protección ofrecida como servizo por un terceiro. Na actualidade, practicamente o monopolio deste servizo está nas mans de Adobe[2] co seu sistema ADEPT (Adobe Digital Editions Protection Technology), que está sendo empregado por multitude de plataformas grandes de venda de *ebooks* e integrado pola maioría de aplicacións de lectura existentes para todo tipo de dispositivos como *eReaders*, *smartphones*, *tablets* ou ordenadores.

Non obstante, a integración da solución DRM de Adobe ten un custo económico, que inclúe un fixo ao inicio e unha porcentaxe de cada venda, que resulta excesivo para a maioría de editoras ou asociacións de editoras que pretenden abordar o desenvolvemento dunha plataforma de distribución de *ebooks* propia. Outras consideracións do emprego de ADEPT son a limitación a uns formatos concretos (EPUB e PDF); a obriga ao lector de rexistrarse como usuario de Adobe, non sendo suficiente para el con ser usuario da plataforma editorial na que merca os *ebooks*; a adecuación do modelo de distribución a uns tipos de permisos concretos, que poden non adaptarse aos requisitos dun proxecto particular; e mesmo a súa vulnerabilidade, pois a pesar de implementar un modelo de protección seguramente moi complexo, e debido precisamente ao seu carácter monopolista, é froito de constantes ataques que aseguran romper esta protección.

Como acabamos de ver, compresión, indexación e protección son tres aspectos

[2] `http://www.adobe.com`

a ter moi en conta na distribución de contidos dixitais. Nas próximas seccións preséntanse diferentes solucións tecnolóxicas que desenvolvemos no marco desta tese para dar resposta a estas tres importantes cuestións no ámbito da distribución de *ebooks*, así como a súa integración en diferentes plataformas comerciais reais. Explicaremos tamén como estas solucións fan uso de estruturas de datos compactas e algoritmos avanzados, resultado da investigación presentada na primeira parte desta tese, obtendo desta forma sistemas cun elevado grado de innovación.

7.1. Solucións para a distribución de *ebooks*

Nesta sección preséntanse diferentes solucións para a distribución de *ebooks* que desenvolvemos e integramos en distintas plataformas comerciais (sección 7.2) no marco desta tese. Estas plataformas presentan características diferenciadas en función do seu contexto de uso, como por exemplo a que usuarios vai dirixida, que formatos de *ebooks* se distribúen, desde que dispositivos se van ler, o modelo de negocio (venda, préstamo etc.), ou as características da conexión á rede, entre outras.

Foi precisamente a observación desta heteroxeneidade a que nos levou a tomar a decisión de abordar o problema xeral da distribución de libros electrónicos separando os distintos aspectos involucrados nel, e desenvolvendo solucións específicas para cada un destes aspectos. O resultado foi que, en lugar de obter unha plataforma única cunhas características moi concretas, obtivemos un conxunto de solucións que se poden combinar e integrar en calquera plataforma de distribución de acordo aos seus requisitos funcionais.

Desde o punto de vista da arquitectura da plataforma, a característica que máis inflúe no seu deseño é o modelo de distribución elixido. Como xa comentamos, existen fundamentalmente dúas modalidades que son, a descarga e a lectura en (*streaming*). Na modalidade de descarga existe, dunha banda, a plataforma de distribución que actúa de servidora dos contidos, e, doutra banda, unha ou varias aplicacións de lectura que poden ser específicas (no caso de que se distribúan os *ebooks* nun formato propio) ou non necesariamente (no caso de que se distribúan arquivos en formatos estándar). Pola contra, na modalidade de *streaming* a propia plataforma de distribución ten que proporcionar as funcionalidades necesarias para a lectura dos *ebooks* a través de Internet, é dicir, ten que integrar un visor *online* propio.

7.1.1. Distribución con descarga

Esta segue a ser na actualidade a modalidade máis empregada polas plataformas de distribución, se ben nos últimos tempos estase a producir un incremento dos modelos que permiten a lectura de *ebooks* na nube (*streaming*). Esta tendencia

vén motivada pola mellora nos dispositivos e tecnoloxías de acceso á Internet que permiten dispor de conexión practicamente sen restricións temporais nin xeográficas.

Un dos aspectos a resolver na distribución de *ebooks* mediante descarga é o de establecer os mecanismos de protección axeitados (DRM) que garantan o acceso aos contidos só por parte dos usuarios autorizados, salvagardando os dereitos de autoría e explotación dos mesmos. Esta protección non é trivial, xa que desde o momento no que un usuario descarga unha copia dixital do libro non hai nada que poida facerse para evitar que a distribúa entre os seus contactos ou de forma indiscriminada a través de Internet.

Outro dos aspectos importantes é o emprego de técnicas que compresión que permitan reducir o tamaño dos *ebooks*, o que repercutirá de forma directa nunha redución do espazo que precisará a plataforma para almacenar o catálogo de publicacións, mais tamén no tempo de descarga para o usuario, e no espazo de almacenamento dos *ebooks* nos dispositivos de lectura. Veremos como o emprego de técnicas propias (resultado da investigación presentada na primeira parte desta tese) que combinan compresión e indexación de texto (autoíndices) permiten non só reducir o espazo de almacenamento dos *ebooks*, senón tamén mellorar os tempos de acceso e busca.

Protección dos *ebooks*

A distribución de contidos dixitais de pago na modalidade de descarga vai precisar protexer, en maior ou menor medida, os *ebooks* distribuídos, a fin de evitar a súa distribución incontrolada. Para este fin, desenvolvemos e presentamos a seguir unha solución para a protección dos libros electrónicos que permite establecer diferentes niveis de DRM.

Dunha banda, desenvolveuse unha solución de DRM lixeiro consistente en incorporar aos *ebooks*, nalgún lugar visible pero non molesto para a lectura, algún dato identificador do seu propietario, como o nome, apelidos e DNI. En realidade, esta modalidade de protección, coñecida como DRM social, non ofrece ningún mecanismo tecnolóxico que controle ou evite a distribución dos *ebooks*, senón que se basea na hipótese de que incorporar os seus datos persoais vai retraer ao comprador á hora de distribuír indiscriminadamente o *ebook*. Esta variante só implica unha leve modificación no propio texto do *ebook* e, polo tanto, permite manter o seu formato orixinal. Para as editoras, esta modalidade presenta a vantaxe de que vai supor, en xeral, un custo menor de integración que unha modalidade de DRM forte. Para o lector, a principal vantaxe é a facilidade de acceso aos *ebooks*, que se manteñen no formato orixinal e poden ser accedidos sen ningún tipo de limitación tecnolóxica. Os formatos soportados nesta modalidade son PDF e EPUB.

Doutra banda, desenvolvemos unha solución de DRM forte que ofrece un alto

nivel de protección aos *ebooks*, e permite garantir que só os usuarios con permiso para facelo poden acceder a eles e nas condicións establecidas (límite de dispositivos, acceso temporal etc.). Foi preciso polo tanto desenvolver as funcionalidades do lado servidor do DRM, responsables de incorporar as autorizacións nos libros distribuídos e controlar o seu cumprimento no momento do acceso, e as funcionalidades do lado cliente integradas nas aplicacións de lectura para determinar se o lector está autorizado para visualizar o *ebook*, e para o que pode precisar comunicarse co servidor. Por este motivo, cada plataforma de distribución que integre esta solución de DRM forte vai ter que proporcionar aos usuarios as súas propias aplicacións de lectura.

A solución desenvolvida é independente do formato de publicación do libro electrónico (*format-agnostic* DRM), e define un formato propio de distribución. No momento da descarga, o contido orixinal é encapsulado neste formato propio, que inclúe unha capa de protección coas autorizacións de acceso a dito contido. No momento da lectura no dispositivo do usuario, a parte cliente do DRM accede a esta capa protectora do contido, comproba se hai autorizacións dispoñibles para o usuario e, de ser o caso, extrae o contido orixinal e envíao ao motor de renderizado da aplicación de lectura. Desta forma conseguimos separar na aplicación de lectura os aspectos de protección (DRM) e visualización do *ebook*, o que permite desenvolver librerías DRM cliente en distintas tecnoloxías (Android, iOS etc.) que se poden integrar en calquera aplicación de lectura para a apertura de contidos protexidos co noso DRM, con independencia dos formatos orixinais que sexa quen logo de reproducir.

No momento de subida á plataforma de distribución, os *ebooks* almacénanse no seu formato orixinal (PDF, EPUB etc.), e é no intre no que un usuario merca un libro e solicita a súa descarga, cando o módulo DRM integrado na plataforma empaqueta o *ebook* no formato de distribución, protexido para o usuario concreto que o mercou. Este proceso de empaquetado segue os seguintes pasos:

1. Creación dunha cabeceira de arquivo que inclúe un número máxico, que permitirá verificar o formato do arquivo, e distinta información de control (como un código de identificación da plataforma de distribución ou os identificadores do *ebook* e do usuario na plataforma) empregada polas aplicacións de lectura para xestionar a comunicación coa parte servidora do DRM integrada na plataforma.

2. Xeración dun arquivo "info.xml" con información do recurso protexido, que inclúe as autorizacións concedidas ao seu propietario (data de caducidade, permiso de impresión etc.), os *metadatos* ou información descritiva do *ebook* (título, autoría, sinopse etc.), o seu formato orixinal (PDF, EPUB etc.), e outra información de xestión como, por exemplo, o número de versión do software co que foi protexido.

3. Cifrado do *ebook* xunto co arquivo con información do recurso protexido creado no paso anterior, empregando un esquema mixto que se explica a continuación. Opcionalmente, pode realizarse previamente algún tipo de procesamento do contido do *ebook* con algunha finalidade concreta, como reducir o seu tamaño (tal e como se verá máis adiante neste mesmo capítulo).

4. Finalmente, concaténase a cabeceira creada no punto 1 co resultado do cifrado no punto 3 para obter o arquivo no formato de distribución final.

Como se indicou no punto 3, para o cifrado do *ebook* emprégase un esquema mixto que combina as técnicas RSA, que é un algoritmo de cifrado asimétrico (ou de clave pública), e AES, que é un algoritmo simétrico (ou de clave secreta). Por eficiencia, resulta conveniente empregar un algoritmo simétrico para o cifrado dos contidos, dada a elevada complexidade computacional dos asimétricos, aínda que os algoritmos simétricos son máis vulnerables no proceso de intercambio ou distribución da clave de cifrado. Empregando un esquema mixto conseguimos explotar as vantaxes de cada un dos métodos, obtendo unha solución eficiente e segura.

Cada usuario deberá ter asociadas na plataforma de distribución un par de claves de cifrado (de lonxitude 2048 bits), que serán a súa clave pública e a súa clave privada para o emprego con RSA, e que se xeran de forma aleatoria no momento en que se dá de alta na plataforma. Cando o usuario instala e utiliza por primeira vez unha aplicación de lectura, esta vaille solicitar as súas credenciais (*login* e *password*) e vainas enviar á plataforma para verificar a súa identidade. No caso de que a autenticación teña éxito, a plataforma enviará de volta á aplicación de lectura a clave privada do usuario, que se almacenará nun lugar seguro de dita aplicación. Para manter a seguridade do sistema é imprescindible que esta comunicación entre a parte cliente do DRM, nas aplicacións de lectura, e a parte servidora, na plataforma de distribución, se realice a través dunha canle segura (normalmente vía HTTPS), para evitar que se poidan producir ataques tipo *man-in-the-middle* que permitan a un terceiro capturar a clave privada dun usuario.

Cando un usuario solicita a descarga dun *ebook* mercado, xérase unha clave aleatoria (de lonxitude 256 bits) que se emprega para cifrar o seu contido, xunto co arquivo con información do recurso protexido, con AES. A continuación, esta clave aleatoria é cifrada á súa vez con RSA empregando a clave pública do usuario. Por último, cando o usuario intenta abrir o *ebook*, a aplicación de lectura empregará a súa clave privada para descifrar a clave aleatoria coa que se cifrou o contido do *ebook* con AES, de forma que só se a clave aleatoria RSA foi cifrada coa clave pública do usuario autenticado na aplicación de lectura, será posible o descifrado e visualización do *ebook*.

Ademais de permitir a protección de *ebooks* en calquera formato de publicación, a solución DRM desenvolvida tamén permite o emprego de diferentes tipos de

autorizacións ou permisos de uso dos libros. Na actualidade temos implementadas diferentes autorizacións e permisos que responden aos requisitos de control sobre o uso dos *ebook* das diferentes plataformas de distribución desenvolvidas:

- Data de caducidade: permite establecer unha data a partir da cal o usuario deixa de ter permiso de lectura sobre o *ebook*. No caso de existir esta restrición de uso, as aplicacións de lectura van precisar facer a comprobación, comparando a data de caducidade coa data actual, antes de abrir o libro. Para impedir que o usuario poida saltar esta restrición manipulando a data de sistema no dispositivo de lectura, é preciso empregar unha fonte de sincronización externa. Neste caso esta fonte de sincronización será a propia plataforma de distribución que enviará a súa data e hora ás aplicacións de lectura en resposta ás peticións que estas realizarán a través da API servidora DRM. Mais esta comprobación remota é demasiado restritiva para o modelo de distribución mediante descarga que, precisamente está pensado para poder funcionar tamén sen conexión a Internet. Por este motivo establécese un mecanismo de dobre comprobación, local e remota, que só vai precisar sincronizar a data coa plataforma en caso de detectar algunha incongruencia na comprobación local, como por exemplo que a data actual do dispositivo sexa anterior á rexistrada do último acceso.
- Tempo de lectura: permite establecer o tempo máximo que o usuario pode ter aberto un *ebook* nun dispositivo de lectura. Esta comprobación faise de forma independente en cada aplicación lectora, que rexistra e acumula o tempo de cada sesión nun contador xeral, e no momento en que se supera o tempo límite, impide que o usuario poida volver abrir o libro.
- Número de dispositivos: permite fixar un límite ao número de aplicacións de lectura que pode instalar un usuario da plataforma de distribución. Esta comprobación faise no servidor DRM cando o usuario inicia por primeira vez unha aplicación de lectura e intenta autenticarse. Por este motivo a aplicación de lectura vai enviar, ademais das credenciais introducidas polo usuario, un identificador do dispositivo (*device ID*) no que está instalada, xunto con outra información descritiva do mesmo (tipo de dispositivo, sistema operativo etc.). O *device ID*, que loxicamente ten que ser único para cada dispositivo, queda rexistrado na plataforma, de modo que é posible controlar o número de dispositivos nos que se instala a aplicación de lectura, e rexeitar novas instalacións cando se supera o límite establecido.
- Permiso de impresión: emprégase para permitir ou limitar a posibilidade de imprimir un *ebook*. Este permiso pode refinarse, por exemplo, permitindo só a impresión dunha porcentaxe dos contidos do libro, ou un número determinado de copias etc.
- Permiso de copia: emprégase para permitir ou limitar a posibilidade de copiar un fragmento do texto do *ebook*. Sería a funcionalidade típica de seleccionar un

texto visible na pantalla e copialo a outra aplicación, como un editor de texto por exemplo. Ao igual que o permiso de impresión, tamén é posible refinar este permiso, por exemplo, fixando un límite ao número de caracteres que se permiten copiar.

A responsabilidade de controlar o cumprimento das autorizacións recae, segundo o caso de que se tratar, na parte servidora do DRM (número de dispositivos), ou na parte cliente (data de caducidade, tempo de lectura, permiso de impresión, permiso de copia). As autorizacións que precisan ser controladas pola parte cliente son codificadas polo servidor e engadidas ao arquivo de información do recurso protexido no momento do empaquetado. Ao ir este arquivo cifrado xunto cos propios contidos do libro, non é posible a súa manipulación para modificar os permisos (sempre e cando non se rompa o cifrado).

Por último, cabe salientar que, ao iren as autorizacións codificadas e protexidas dentro do propio contido distribuído (no formato propio de *ebook*) e ao recaer a responsabilidade sobre o seu control exclusivamente en compoñentes da propia solución DRM (parte servidora e/ou cliente), vai ser posible introducir de forma sinxela novos tipos de autorizacións. Isto permite que a solución DRM desenvolvida non só dea soporte ás necesidades das plataformas actuais que incorporan modelos de negocio coñecidos como a venda individual, a subscrición ou o préstamo, senón tamén á aparición de novos modelos no futuro.

Compresión e indexación do texto

Os libros electrónicos están constituídos fundamentalmente por texto en linguaxe natural e outros contidos multimedia, como imaxes ou vídeos. Se ben estes últimos se atopan sempre en formatos que empregan técnicas de compresión específicas para reducir o seu tamaño (como JPG, PNG ou GIF para imaxes; MP4, WebM ou OGG para vídeos etc.), non ocorre o mesmo cos contidos textuais. Algúns formatos de distribución inclúen na súa especificación o emprego dalgunha técnica de compresión, como por exemplo o formato de publicación EPUB que utiliza Deflate, para reducir o espazo de todos os arquivos que conforman a publicación, incluídos os textos. Debido á necesidade de comprimir información de distinta natureza (texto, imaxes, vídeos etc.) utilízanse técnicas xerais de compresión de datos orientadas a bytes que, a diferenza das técnicas de compresión específicas, non permiten aproveitar as características do texto en linguaxe natural para optimizar a compresión nin incorporar funcionalidades como o acceso directo a posicións no texto. Por exemplo, a descompresión con Deflate é secuencial (byte a byte), polo que se unha aplicación de lectura precisa acceder a unha posición arbitraria dun texto, por exemplo debido a un avance rápido de páxina ou ao acceso a un marcador prefixado polo usuario, de forma xeral vai ter que descomprimir o texto desde o principio.

Doutra banda, para ofrecer ao usuario a posibilidade de realizar procuras no texto completo do libro, a aplicación de lectura vai precisar acceder a todos os arquivos de texto que forman o libro, descomprimilos (aqueles que non teña nese momento xa descomprimidos en memoria), e facer un recorrido secuencial por eles para localizar a cadea buscada. Outra opción sería manter un índice á parte para resolver a procura, por exemplo un índice invertido, e logo descomprimir só os arquivos de texto que conteñen algún resultado, e só até o punto no que aparece a cadea buscada. Esta opción implicaría un incremento do espazo de almacenamento necesario no dispositivo de lectura, e tamén un maior procesamento para construír o índice no momento de incorporar o *ebook*. Probablemente por todo isto, moi poucas aplicacións ofrecen a posibilidade de buscar sobre o texto completo dos *ebooks*.

Neste punto, cabe sinalar como unha das estruturas que presentamos como contribución desta tese, o `WCSA` (Word-Based CSA) 4, encaixa á perfección para o obxectivo de representar de forma comprimida o texto dos *ebooks*. Son tres as características do `WCSA` que o fan axeitado para o seu emprego neste contexto:

1. Emprega unha técnica de compresión específica para texto en linguaxe natural que permite unha compresión moi eficiente, e que ademais se pode axustar mediante parametrización do algoritmo para obter distintos niveis de compresión que influirán tamén no tempo de acceso (balance espazo-temporal).

2. Permite realizar procuras moi eficientes sobre o texto completo, ao tratarse tamén dunha técnica de indexación (autoíndice).

3. Ofrece acceso directo a calquera posición (palabra) do texto para comezar a descomprimir desde ela. Isto permite manter o texto comprimido en memoria principal e ir descomprimindo só o fragmento que se precisa nun momento dado, ofrecendo así un rendemento óptimo tanto para lectura secuencial como para a realización de saltos a posicións arbitrarias.

Estas características fan moi interesante o emprego de `WCSA` para comprimir os contidos textuais dos *ebooks* que se distribúen na modalidade de distribución con descarga. A compresión ten lugar na plataforma no momento do empaquetado do *ebook* no formato de distribución e xusto antes do cifrado, mentres que a descompresión ten lugar nas aplicacións de lectura na orde inversa, isto é, xusto despois do descifrado.

Polo xeral, o formato orixinal dos *ebooks* empregará algún tipo de arquivo contedor (*single-file container*), polo que para realizar a compresión será preciso extraer previamente os arquivos que forman parte da publicación orixinal. Loxicamente, esta operación vai depender de cal sexa o procedemento de empaquetado empregado, e de que se dispoña da súa especificación para poder interpretalo. Na actualidade, e debido ao seu emprego nas plataformas que desenvolvemos, temos implementados

módulos para a extracción dos arquivos de recursos empaquetados en ZIP[3] e en EPUB, que emprega a especificación EPUB Open Container Format (OCF)[4].

Unha vez obtidos de forma illada os arquivos que conforman o *ebook*, clasifícanse en arquivos de texto, que polo xeral empregarán algunha linguaxe de marcas como HTML ou XML, e binarios, que serían todos os demais e que poderían representar información de distinto tipo, como imaxes, vídeos, audios, *clips* de animación Adobe Flash[5] etc. Isto permítenos comprimir os arquivos de texto con `WCSA`, deixando os binarios no seu formato orixinal pois, como xa comentamos, estes van estar xa en formatos comprimidos con técnicas específicas, polo que aplicar calquera outra técnica de compresión sobre eles vai penalizar o tempo de acceso e probablemente non reducirá (nalgunhas probas que fixemos, mesmo aumentaba) o espazo ocupado por eles.

Co obxectivo de incrementar a redundancia do texto e obter así un mellor factor de compresión, decidimos concatenar previamente todos os arquivos textuais e comprimir o texto resultante, en lugar de facelo con cada arquivo por separado. Isto é posible grazas a que `WCSA` ofrece acceso directo a calquera posición do texto, polo que basta con manter unha táboa co nome de cada arquivo e a posición onde empeza dentro do texto concatenado comprimido para poder acceder de forma eficiente a el. Notar que, se ben a concatenación dos contidos leva implícita unha ordenación, esta podería ser totalmente independente da orde na que os contidos se acceden no momento da lectura, pois ao proporcionar `WCSA` acceso directo eficiente a calquera posición do texto fai que esta solución ofreza un acceso tamén eficiente tanto para os contidos de lectura fundamentalmente secuencial como para aqueles que seguen un patrón de acceso arbitrario a través de hiperenlaces.

Como acabamos de ver, o emprego de `WCSA` para a compresión do texto dun *ebook* permite reducir o seu espazo de almacenamento e realizar buscas moi eficientes sobre el, mais para poder resolver eficientemente a busca sobre o texto de todos os *ebooks* dispoñibles no catálogo da plataforma, é preciso dispor de algunha estrutura adicional que evite ter que buscar de forma individual no texto de cada un dos libros. A continuación amósanse dúas posibles alternativas que fan emprego de índices invertidos como estruturas auxiliares para resolver esta cuestión:

1. Empregar un índice invertido a documentos (os *ebooks*). Na figura 7.2 amósase un esquema desta aproximación, na que se constrúe un índice invertido con todas as palabras que aparecen algunha vez no texto dos *ebooks*, e de cada unha delas a listaxe de *ebooks* nos que aparece.

   O índice invertido permite resolver a busca dos *ebooks* que conteñen unha palabra (*document retrieval*), mais para poder ofrecer a funcionalidade de

[3] `https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT`
[4] `http://www.idpf.org/epub/301/spec/epub-ocf.html`
[5] http://www.adobe.com/es/products/flash.html

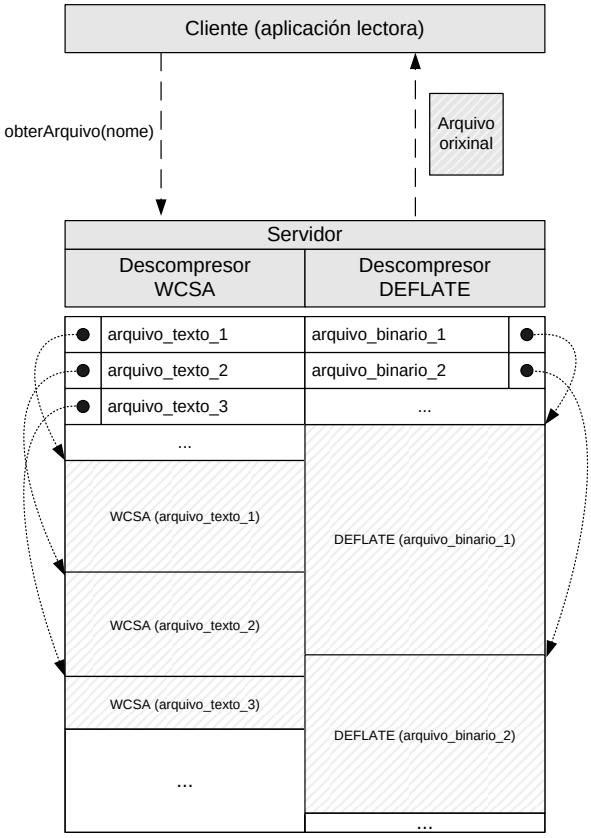

Figura 7.1: Empaquetado do formato de distribución propio.

busca sobre o texto completo, isto é, saber en que posición exacta aparece unha palabra ou frase no texto dos libros, o índice invertido a documentos resulta insuficiente. Non obstante, o feito de ter comprimido o texto de cada *ebook* cun autoíndice como o `WCSA` permite resolver a operación de busca en tempo completo de forma eficiente en dúas fases. Na primeira fase, resólvese a busca dos *ebooks* que conteñen a palabra ou conxunto de palabras a través do índice invertido, e na segunda efectúase a busca de texto completo en cada un dos *ebooks* recuperados na busca anterior, mediante as funcionalidades de busca que proporciona `WCSA`.

2. Empregar un índice invertido que apunte a posicións exactas no texto dos *ebooks*, que resulta suficiente para resolver por si mesma as procuras de texto

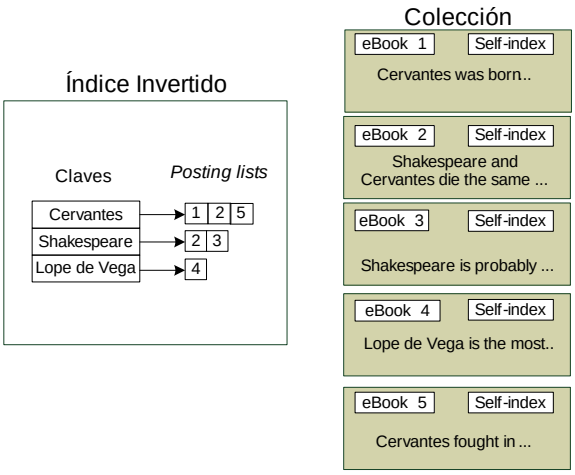

Figura 7.2: Solución para busca no servidor con índice invertido a documentos.

completo mediante intersección de listas no índice invertido.

Aínda que o feito de que `WCSA` proporcione acceso directo na descompresión permite aforrar certo espazo ao gardar nas *posting list* as posicións das palabras na representación comprimida do texto, en lugar de posicións no texto plano, o certo é que esta solución precisa unha cantidade de espazo considerablemente superior, habitualmente entre 3 e 5 veces maior, á que emprega un índice invertido orientado a documentos como o explicado no punto anterior.

Como resumo, o emprego de `WCSA` para a representación comprimida do texto dos *ebooks* ofrece unha serie de vantaxes como son:

- Optimización da compresión dos contidos textuais dos *ebooks*, mediante a aplicación dunha técnica específica para texto en linguaxe natural e a explotación da redundancia conxunta.
- Control do balance espazo-temporal mediante a parametrización do algoritmo, o que permite optar por unha solución máis eficiente en canto a espazo requirido ou tempo de acceso segundo sexa preciso.
- Favorece a posibilidade de manter o libro completo en memoria principal do dispositivo lector, ao reducir o seu tamaño, o que supón unha mellora significativa no tempo de acceso aos contidos con respecto a ter que ir buscalos a disco.

- Permite realizar buscas eficientes no texto completo do *ebook* sen necesidade de empregar estruturas de indexación auxiliares.
- Permite desenvolver unha solución moi eficiente en canto a espazo para a busca de texto completo sobre todo o catálogo de *ebooks*, mediante o emprego dun índice invertido a documentos auxiliar.

Por último, salientar que aínda que o cifrado e compresión dos contidos se van combinar normalmente para a distribución dos *ebooks*, as dúas operacións serían opcionais e podería desenvolverse unha solución de distribución na que só se emprega o cifrado para a protección dos *ebooks* ou unha na que só se emprega a compresión para reducir o seu tamaño. Cabe salientar neste punto o feito de que o emprego de estruturas de datos complexas como as do `WCSA`, fai que, salvo que se coñezan a fondo ditas estruturas, os algoritmos necesarios para descodificalas e os parámetros concretos cos que foron construídas (no caso de que sexan parametrizables), é realmente complexo poder reconstruír o texto orixinal a partir delas. Se ben resulta evidente que non ofrece a seguridade dun cifrado, si introduce un nivel de ofuscamento que pode resultar suficiente en moitos contextos nos que, polo tanto, sería posible prescindir do cifrado, coa vantaxe que isto suporía en canto á eficiencia da solución desenvolvida. Isto que se presenta aquí como *proba de concepto*, permitiría abrir unha liña de investigación no emprego de estruturas de datos compactas deseñadas para a compresión e indexación de texto, co propósito adicional de que introduzan certo nivel de protección ou ofuscamento, que dificulte a reconstrución do texto orixinal para quen non dispoña de determinada información empregada no proceso de creación das estruturas.

7.1.2. Distribución en *streaming*

Cada vez existe un maior número de plataformas de distribución de *ebooks* que ofrecen acceso á lectura a través de visores, integrados na propia web da plataforma ou como aplicacións independentes, que van descargando os contidos do libro directamente de Internet a medida que o lector avanza por el (*streaming*). A consolidación deste modelo de distribución afecta non só a plataformas de libros electrónicos (Kindle Cloud Reader, Google Books, Kobo Instant Reader, 24Symbols, Nubico etc.), senón tamén a outras que distribúen vídeo baixo demanda (Netflix, Wuaki, Yomvi etc.) ou música (Spotify, Google Play Music, iTunes Radio, Grooveshark etc.).

A diferenza das plataformas que distribúen os libros electrónicos mediante descarga, na modalidade de *streaming* o usuario non dispón en ningún momento dunha copia dixital completa do *ebook* no seu dispositivo, senón que vai recibindo fragmentos do seu contido a medida que avanza na lectura. En determinados

contextos, esta distribución fragmentada baixo demanda representa por si mesma un nivel de protección considerado suficiente[6], xa que dificulta a posibilidade de que o usuario poida reconstruír o libro completo no seu formato orixinal. É habitual tamén que as aplicacións de lectura, sexan web ou nativas, incorporen algunhas funcionalidades dirixidas a dificultar na medida do posible a extracción dos contidos, como por exemplo limitar ou inhabilitar determinadas opcións como a de copia ou impresión do texto.

En todo caso, o feito de que o usuario teña que estar conectado en todo momento durante a lectura permite introducir no servidor de *streaming* un certo nivel de protección a través do control na distribución dos contidos. Así, se na modalidade con descarga só é posible controlar a distribución dun *ebook* no momento da descarga e sobre o contido completo do libro, no modelo de *streaming* é posible facer o control en tempo real durante a lectura e a nivel de fragmento. Isto é, ante cada solicitude de novo fragmento a plataforma vai ter que verificar a identidade do usuario solicitante, comprobar os seus permisos e, consecuentemente, aceptar ou denegar o envío. Este control en tempo real e fraccionado sobre a distribución dos contidos facilita a integración de diferentes modelos de negocio, como os baseados no control do tempo de acceso (subscrición, aluguer etc.) ou a lectura só desde determinadas familias de dispositivos. En base aos modelos de negocio considerados, que darán lugar a distintos tipos de permisos, sería preciso deseñar a base de datos de autorizacións para poder almacenar nela os permisos correspondentes, e implementar o seu control no servidor de *streaming* que, nalgúns casos, pode implicar tamén que as aplicacións de lectura envíen algunha información adicional nas súas peticións.

A continuación preséntase unha solución desenvolvida para a distribución de *ebooks* en formato EPUB mediante *streaming* que temos integrada en distintas plataformas, e tamén nunha tenda *online* de *ebooks* (Xcloud Bookstore[7]) que Enxenio comercializa como produto para editoras. A arquitectura desta solución está constituída polo módulo Servidor de *Streaming* responsable de controlar o envío dos contidos do *ebook* aos usuarios autorizados, a Base de Datos de Autorizacións que almacena os permisos de acceso dos usuarios aos *ebooks*, o Repositorio de *Streaming* onde se almacenarán os contidos dos *ebooks*, e unha Aplicación Lectora desenvolvida con tecnoloxías web de cliente (HTML5), encargada de realizar as peticións ao módulo servidor, en resposta á actividade do usuario lector, e representar os contidos devoltos por este no visor.

[6]De requirir un nivel de protección maior sería preciso desenvolver unha modalidade forte de DRM mediante o emprego dalgún método criptográfico para o envío cifrado dos contidos. Nese caso, un aspecto moi importante a considerar vai ser o emprego de técnicas que ofrezan unha operación de descifrado eficiente e lixeira computacionalmente, especialmente no caso dos visores web que utilicen tecnoloxías de cliente dos navegadores (*javascript*) para realizar dita operación.

[7]`http://www.xcloud-bookstore.com`

Servidor de *Streaming*

O servidor de *streaming* é o compoñente encargado de controlar a distribución dos contidos dos *ebook* ás aplicacións de lectura para o seu acceso por parte dos usuarios autorizados. No momento da subida dos *ebooks* á plataforma, o servidor de *streaming* desempaqueta o EPUB e almacena os recursos básicos de contido no repositorio de *streaming*. Isto faise así por unha cuestión de eficiencia, xa que ao dispor dos recursos de contido por separado, o servidor de *streaming* vai poder acceder máis rápido a eles que no caso de manter o *ebook* empaquetado no seu formato orixinal, e responder así máis eficientemente ás peticións das aplicacións de lectura e reducir o seu tempo de resposta aos usuarios.

Para o desenvolvemento da nosa solución para a distribución de *ebooks* en *streaming* decidimos resolver o aspecto da protección a través do control na distribución fragmentada dos contidos, o que nos permite ofrecer unha solución eficiente (ao non requirir descifrado dos contidos) e cun nivel de protección que resultou suficiente para o seu emprego nas distintas plataformas na que se empregou. Na actualidade temos desenvolvidos o modelo de venda individual dos *ebooks*, coa autorización de acceso ilimitado para o usuario comprador, e o de subscrición ao catálogo completo de libros dispoñibles na plataforma, coa autorización de data de caducidade do acceso á lectura.

Visor web *streaming* (Xcloud Reader)

Na parte cliente da arquitectura da modalidade de distribución en *streaming* atópanse as aplicacións de lectura, desde as que os usuarios teñen acceso aos contidos dos *ebooks*, e que se encargan da comunicación co servidor de *streaming* para ir descargando baixo demanda ditos contidos. Debido á gran diversidade dos dispositivos desde os que é posible acceder hoxe en día á lectura de libros na nube (ordenadores, *tablets*, *smartphones* etc.), resultaría moi custoso desenvolver unha aplicación de lectura nativa para cada un destes dispositivos e tecnoloxías, polo que a nosa aposta inicial foi desenvolver unha aplicación lectora baseada no emprego de tecnoloxías web, á que chamamos Xcloud Reader, que ofrece unha solución practicamente universal ao permitir o acceso á lectura desde calquera dispositivo que dispoña dun navegador web máis ou menos actual. Ademais, a interface de usuario foi especialmente deseñada para axustarse ao tamaño de pantalla do dispositivo (*responsive design*), ofrecendo unha boa experiencia de usuario na lectura tanto en pantallas grandes como nas de un *smartphone*.

En canto ás funcionalidades que Xcloud Reader ofrece ao usuario durante a lectura, son semellantes ás que ofrecen a día de hoxe as aplicacións máis empregadas para a lectura de *ebooks* como:

- Navegación: ofrece a posibilidade de avanzar e retroceder páxina a páxina, mediante botóns na interface de escritorio ou co xesto de deslizar (*swipe*) na interface táctil, e tamén o avance rápido (*scrolling*) polo contido do *ebook*.

- Índice: a partir da táboa de contidos (*table of contents*, TOC) do EPUB, xérase un índice que permite o acceso directo ás diferentes seccións do *ebook*.

- Ficha descritiva: ofrece vista en modo de ficha dos campos descritivos (*metadatos*) do libro, como a imaxe de portada, o título, a autoría ou a sinopse. Segundo a especificación de EPUB para a definición destes campos emprégase o conxunto de *metadatos Dublin Core Metadata Element Set*[8].

- Preferencias de lectura: permítese a configuración de diferentes aspectos relacionados coa representación do contido para axustalos ás preferencias de lectura do usuario, como o estilo da fonte, o seu tamaño, o espazo entre liñas e a combinación entre a cor de fondo e do texto.

Aínda que Xcloud Reader foi desenvolvido para ofrecer acceso a lectura con conexión a Internet mediante *streaming*, desde un principio consideramos que en determinados contextos resultaría moi útil dispor da posibilidade de continuar a lectura dun *ebook* ante perdas temporais da conexión. O exemplo típico sería o da lectura desde dispositivos móbiles conectados a Internet a través de redes de telefonía, nos que as perdas puntuais de conexión debido a problemas de cobertura poden ser relativamente frecuentes. Para resolver esta cuestión, empréganse dúas características definidas pola última revisión da linguaxe HTML, denominada HTML5, que son *offline web applications* e *local storage*:

- *Offline web application* define un mecanismo (*cache manifest*) para que unha web declare que determinados recursos (HTML, CSS, JavaScript, imaxes etc.) deberán ser almacenados na caché no navegador para que estean logo dispoñibles sen conexión. Se posteriormente se accede á dita web sen conexión, no canto de obter unha mensaxe de erro, vanse cargar da caché os recursos que se almacenaron previamente con conexión. No caso de Xcloud Reader almacénanse tanto os HTML e CSS que representan a interface do lector como os arquivos Javascript co código das diferentes funcionalidades de lectura e acceso aos contidos do *ebook*.

- *Local storage* define un mecanismo para almacenar datos (un mínimo de 5MB) de forma persistente no navegador web. Estes datos serán pares nome/valor de cadeas de caracteres (tipo *string*). Xcloud Reader emprega este mecanismo para almacenar os contidos do *ebook* e poder acceder a eles sen conexión.

[8] `http://dublincore.org/documents/dces/`

Así, baseándonos nestas dúas características para as que a maioría de navegadores ofrecen soporte hoxe en día, desenvolvéronse dúas funcionalidades diferentes para permitir a lectura *offline*:

- Descarga automática predictiva: a medida que o usuario avanza pola lectura do libro, Xcloud Reader vai solicitando de forma automática fragmentos de contido que irían a continuación, nunha lectura secuencial, ao que está a amosar nese momento. Estes fragmentos son almacenados no navegador (*local storage*) e desde aí son cargados cando se precisan. No caso de perda temporal da conexión, o usuario vai poder seguir lendo, sen decatarse, até que chegue ao final dos fragmentos almacenados no navegador. Nese momento, se xa se recuperou a conexión, a caída puntual sería completamente transparente para o usuario e non afectaría á súa lectura.

- Descarga manual completa: neste caso, é o usuario o que solicita a Xcloud Reader a descarga completa dos fragmentos do libro no seu espazo de almacenamento local, o que lle permitirá ler o libro sen restricións de conexión á rede. Dado que o espazo de almacenamento local dos navegadores é limitado, pode darse o caso de que o tamaño do libro supere ao espazo dispoñible nese momento, e polo tanto non sexa posible esta operación.

É obvio que o visor ten que comprobar en todo momento a dispoñibilidade ou non de conexión á rede e, en base a iso, decidir se debe resolver as peticións do usuario comunicándose co servidor de *streaming* ou localmente co navegador. Isto pode dar lugar a inconsistencias entre a información do estado de lectura almacenada polo servidor e a que se almacena localmente no navegador. Por exemplo, se un usuario continúa lendo logo de perder a conexión, o seu progreso real vai quedar rexistrado no almacenamento local do navegador, pero non no servidor. Para minimizar o seu impacto, cada vez que Xcloud Reader detecta que recuperou a conexión, envía ao servidor a información de estado que ten almacenada localmente, e que inclúe unha marca de tempo de cando dita información foi rexistrada. O servidor comproba esta información e, no caso de ser máis recente que a que ten na súa base de datos, actualízaa.

Para rematar, indicar que este visor foi deseñado e desenvolvido como un compoñente illado que podería ser integrado en calquera plataforma web de distribución de libros electrónicos que non empregue o noso módulo servidor de *streaming*. As peticións que Xcloud Reader realiza en resposta ás accións do usuario lector, é polo tanto ás que vai ter que responder calquera plataforma que o integre, descríbense[9] de seguido:

[9]Por claridade, omítense os detalles da especificación da API de integración.

1. Obter estrutura navegación: o visor pídelle ao servidor a estrutura de navegación dun *ebook*, o que lle vai permitir determinar a secuencia dos contidos, xestionar as solicitudes de fragmentos e construír o índice.
   Entrada: o identificador do libro.
   Saída: a estrutura de navegación segundo a especificación do estándar EPUB.

2. Obter *metadatos ebook*: o visor pídelle ao servidor os *metadatos* descritivos do *ebook*.
   Entrada: o identificador do libro.
   Saída: os atributos do libro en Dublin Core.

3. Obter fragmento: o visor solicita o envío dun fragmento do contido do *ebook* (que tería que corresponder con algún dos fragmentos nos que se organiza o contido, resultado da petición 1).
   Entrada: o identificador do *ebook* e o identificador do fragmento.
   Saída: o fragmento en formato XHTML.

4. Obter estado: o visor solicita as preferencias de lectura do usuario e a posición (progreso) na que deixou o *ebook* no seu último acceso. Isto vaille permitir a apertura do *ebook* no punto no que o usuario o deixou na última lectura, e mantendo as súas preferencias de visualización. O visor almacenará localmente (no *localStorage* do navegador) o estado recibido, para poder empregar dita información no caso de que non dispoña de conexión co servidor a seguinte vez que o usuario accede á lectura e non poida actualizala.
   Entrada: o identificador do usuario e o identificador do libro.
   Saída: o estado (preferencias máis progreso) do usuario en formato JSON.

5. Gardar preferencias: o visor informa do cambio na configuración das preferencias de lectura do usuario.
   Entrada: o identificador do usuario e máis as súas novas preferencias.

6. Gardar posición lectura: o visor informa da posición de lectura cada vez que o usuario avanza no contido (cambio de páxina ou avance rápido).
   Entrada: o identificador do usuario, o identificador do libro e o capítulo/posición actual.

Distribución en *streaming* con compresión

A distribución fragmentada dos contidos dos *ebook* mediante *streaming* permite reducir o ancho de banda nas transmisións e os tempos de resposta ofrecidos ao usuario para a lectura. Non obstante, en ocasións poden existir condicionantes tecnolóxicos que fagan preciso considerar o envío dos contidos non só fragmentados, senón tamén comprimidos para reducir aínda máis o tamaño da información transmitida pola rede. Este é o caso de *Vía Inteligente*, un proxecto no ámbito

das *Smart Cities* no que se ofertan distintos servizos para o consumo de contidos dixitais a través de redes sen fíos e servidores integrados no pavimento, e que dispón dunha limitación de ancho de banda de 256Kbps[10].

Un dos servizos que desenvolvemos no marco deste proxecto foi a Viateca, que é unha das plataformas de distribución que se explican máis adiante neste mesmo capítulo (sección 7.2.4), e que ofrece acceso a un catálogo gratuíto de *ebooks* a través da infraestrutura da *Vía Inteligente*. A fin de poder ofrecer unha boa experiencia de lectura aos usuarios a pesar da limitación a 256Kbps no ancho de banda dispoñible para as transmisións, foi preciso desenvolver unha solución para a distribución dos contidos comprimidos. A decisión que tomamos foi a de empregar unha técnica de compresión adaptativa para a transmisión da información textual baseada no algoritmo DLETDC [BFNP10] introducido na primeira parte desta tese (sección 3.2), e que permite a transmisión de texto comprimido en tempo real.

Ademais, outros motivos que motivaron a elección desta técnica foron a porcentaxe de compresión que acada (até un 30–35 % do espazo ocupado polo texto orixinal), e o feito de ofrecer un mecanismo de descompresión moi lixeiro que reduce o procesamento necesario no lado do receptor. Isto é especialmente útil no contexto da *Vía Inteligente*, onde os usuarios acceden aos servizos normalmente a través de dispositivos móbiles, algúns dos cales poden ofrecer unha capacidade de procesamento limitada. Esta redución do procesamento no receptor permite, loxicamente, reducir o tempo de resposta da aplicación lectora, mais tamén outras vantaxes como un menor consumo de batería, algo que na actualidade resulta de vital importancia.

Esta redución do procesamento necesario para a descompresión acádase evitando que o receptor teña que inferir o modelo do texto durante dita operación, sendo o emisor (o servidor de *streaming*) o encargado de transmitir, ademais dos códigos que representan o texto comprimido, os cambios que se producen en dito modelo, e que son calculados dinamicamente.

Sobre a versión orixinal do algoritmo DLETDC fixemos unha modificación para engadir un vocabulario estático coas 128 palabras con maior probabilidade de seren transmitidas, para o que nos baseamos nas estatísticas sobre cales son as formas máis frecuentes[11] do español publicadas pola Real Academia Española. A estas palabras asígnanselle códigos de 1 byte (do 128-255), e representan polo tanto o primeiro nivel da codificación ETDC. O resto do vocabulario constrúese de forma dinámica en base á frecuencia real de aparición das palabras no texto a transmitir, para o que se empregan os códigos a partir do segundo nivel de ETDC e que precisan como mínimo 2 bytes para a súa representación.

[10] Trátase dunha restrición legal que a *Comisión Nacional de los Mercados y la Competencia* (CNMC) establece para a oferta de servizos gratuítos a través de redes wifi, a fin de protexer o mercado dos operadores de Internet móbil

[11] `http://corpus.rae.es/frec/1000_formas.TXT`

A idea detrás do emprego deste vocabulario estático é explotar o modelo probabilístico do idioma para reducir o número de substitucións no vocabulario dinámico, e mellorar aínda máis a eficiencia da operación de descompresión, á vez que reducir tamén o espazo ocupado polo texto comprimido ao incluír un número menor de códigos de substitución. Isto será máis certo canto máis se parezan o modelo probabilístico do texto a comprimir e o do idioma español que tomamos como referencia para o cálculo do vocabulario estático. No caso de que estes modelos sexan moi diferentes, isto é, as palabras máis frecuentes do español aparecen poucas veces no texto a transmitir, o factor de compresión vaise ver prexudicado pola utilización de poucos códigos de 1 byte correspondentes ao vocabulario estático.

Por último, indicar que o emprego desta técnica permite realizar operacións de busca sobre o texto comprimido de forma eficiente, tanto en velocidade (hasta 7 veces máis rápido que a busca sobre texto plano) como en uso de memoria.

7.2. Casos prácticos

Unha vez explicadas as diferentes solucións deseñadas e desenvolvidas para a distribución de contidos dixitais, nesta sección preséntanse unha serie de proxectos concretos nos que se desenvolveron diferentes plataformas de distribución de libros electrónicos, e que polo tanto, integran algunhas das solucións explicadas anteriormente.

7.2.1. Elibro-galego

En 2010, a *Asociación Galega de Editores*[12] (AGE) decide abordar un ambicioso proxecto para ofrecer aos seus asociados un espazo web desde o que comercializar os seus contidos en formato dixital. O resultado foi Elibro-galego (`http://www.elibro-galego.com`), unha plataforma multieditorial e multiformato que ofrece *ebooks* para a súa lectura na nube (*streaming*) ou mediante descarga.

O principal problema co que nos atopamos no desenvolvemento desta plataforma foi atopar unha solución que permitise a distribución de toda a variedade de formatos nos que as máis de 40 editoras da AGE tiñan os seus contidos dixitais (PDF, EPUB, HTML, aplicacións de escritorio, aplicacións móbiles etc.), así como dar soporte e ofrecer os mecanismos de protección adecuados para os diferentes modelos de negocio propostos por cada unha delas. Finalmente as alternativas que Elibro-galego soporta para a distribución de *ebooks* son:

- Distribución con descarga

[12] `http://www.editoresgalegos.org`

- DRM social: emprégase a solución de marcado con nome, apelidos e DNI do comprador presentada na sección 7.1.1 para os seguintes formatos:
  - PDF
  - EPUB
- Formato Elibro-galego: emprégase un formato propio de distribución coa solución de DRM forte explicada na sección 7.1.1 e a compresión con `WCSA` explicada na sección 7.1.1. Os formatos orixinais soportados, e que se empaquetan no formato Elibro-galego para a súa distribución, son:
  - PDF
  - EPUB
  - HTML: denomínase así a un formato de contido visualizable nun navegador web, e que polo tanto pode estar constituído por un conxunto de recursos HTML, CSS, imaxes, vídeos etc. que se empaquetan nun único arquivo ZIP. A única restrición é que estes contidos deben conter un arquivo "index.html" no raíz do ZIP, que será o que se cargue inicialmente ao abrir un contido neste formato.

  Para a visualización dos *ebooks* neste formato desenvolvéronse aplicacións de lectura de escritorio para os sistemas operativos Windows, Linux e Mac OS X.

- Distribución en *streaming*: desenvolveuse un visor web integrado na plataforma que ofrece acceso *online* aos contidos do *ebook* a medida que o usuario avanza pola súa lectura, sendo este control sobre a distribución o mecanismo de protección que inclúe. Pode considerarse como o precursor de Xcloud Reader (sección 7.1.2), mais as funcionalidades que ofrece son moito máis limitadas, non permitindo por exemplo a lectura *offline* nin axustar o texto ao tamaño de pantalla dos distintos dispositivos. Os formatos que soporta son:
  - PDF: o visor permite navegar polas páxinas do PDF de forma secuencial ou con acceso directo a unha páxina concreta. Para evitar a descarga completa do PDF, en realidade cada páxina é unha imaxe en formato JPG xerada no momento de subida do PDF á plataforma.
  - EPUB: o visor permite avanzar, de forma secuencial ou con acceso directo a través do índice de navegación do EPUB, polas diferentes seccións de contido do *ebook*.

En canto aos modelos de negocio implementados, son a venda sen limitación temporal de uso para os *ebooks* ofrecidos para descarga, e a venda dun acceso tamén ilimitado en tempo (non simultáneo) dos ofrecidos a través de *streaming*. Na modalidade de descarga, as únicas restricións que hai son no número de instalacións da aplicación de lectura que pode ter activas un usuario (até un máximo de 5) e

o tempo no que o contido mercado permanece dispoñible para a súa descarga no servidor (30 días). Na modalidade de *streaming*, os *ebooks* mercados permanecen á disposición do usuario mentres manteña a súa conta en Elibro-galego.

Ademais dos formatos de *ebooks* que Elibro-galego distribúe mediante descarga ou *streaming*, a plataforma ofrece tamén soporte á venda doutros produtos dixitais, como aplicacións para PC ou móbiles Java (desenvolveuse unha librería DRM para integrar en aplicacións J2ME), e á promoción doutros formatos non distribuídos nela: ligazóns a puntos de venda para libro físico, ligazón ao *ebook* en formato PDF ou EPUB con DRM de Adobe noutras plataformas de venda, e ligazón ás *apps* móbiles Android e iOS publicadas en Google Play ou Apple Store respectivamente.

7.2.2. Aula Virtual de Baía Edicións

Tamén en 2010, coincidindo temporalmente cos inicios do proxecto Elibro-galego, Baía Edicións[13], unha das editoriais integrantes da Asociación Galega de Editoras con maior tradición en libro de texto educativo, afronta un proxecto para desenvolver unha versión dixital e interactiva do seu libro de Sociais para 1º da ESO (pode accederse de forma gratuíta ao primeiro tema do libro en `http://sociais1.baiaedicions.net`), converténdose así en pioneira entre as editoras de ámbito galego na aposta polo libro de texto virtual.

O proxecto de libro virtual para 1º de Sociais de Baía Edicións non consistiu nunha simple transformación a dixital dos contidos do libro en papel, senón que un dos obxectivos básicos era proporcionar unha serie de actividades interactivas e exercicios de autoavaliación que permitisen sacar partido ás posibilidades do emprego das novas tecnoloxías nas aulas, sendo precisamente esta interacción co libro virtual o elemento diferencial con respecto ao libro en papel. Debido a isto, para o desenvolvemento deste proxecto contamos cun equipo multidisciplinar formado por persoal informático e outro de contidos que, valéndose dunha serie de ferramentas software deseñadas e desenvolvidas polos primeiros, se encargaron de introducir os contidos do libro (textos, imaxes, vídeos, animacións Flash, exercicios etc.) nunha base de datos desde a que esas mesmas ferramentas permitían exportar ao formato final de publicación.

A aceptación do libro virtual de 1º de Sociais de Baía Edicións foi moi boa entre a comunidade educativa. A rede de comerciais da editora distribuíu o libro virtual a máis de 150 profesores de secundaria de 70 centros educativos galegos, que tiveron a oportunidade de probalo e empregalo cos seus alumnos nas súas clases. Esta boa acollida animou a Baía Edicións a continuar o proceso de edición en dixital de máis libros de texto do seu catálogo e outros contidos curriculares, polo que pronto se

[13] `http://www.baiaedicions.net`

decatou da necesidade de dispor dunha plataforma que lle permitise distribuír de forma controlada estes contidos dixitais a través de Internet.

Con este obxectivo, en 2012 encárgasenos o desenvolvemento da Aula Virtual de Baía (`http://aulavirtual.baiaedicions.net`), unha plataforma de distribución controlada de libros de texto virtuais en formato HTML (o mesmo que se detallou na sección 7.2.1). A modalidade de distribución que ofrece é a de descarga dos contidos nun formato propio protexido, para o que integramos na plataforma tanto a solución de DRM forte (sección 7.1.1) como a da compresión dos contidos textuais (sección 7.1.1).

En canto ao modelo de negocio, a decisión que toma Baía Edicións é a de ofrecer de forma gratuíta o libro virtual a aqueles profesores que empreguen o libro de texto en papel, dándolle permiso de uso tanto a eles como aos seus alumnos e durante todo o ano académico. Así pois, o DRM ten que, dunha banda, impedir o acceso ao libro virtual máis alá do fin do ano académico e, doutra banda, controlar o acceso ao libro virtual por parte dun número limitado de usuarios anónimos, pero autorizados (o profesor e os seus alumnos). Para isto, desenvolveuse un mecanismo de control baseado no emprego de códigos de activación, que son cadeas aleatorias de 15 caracteres alfanuméricos que se asocian, na base de datos de autorizacións do servidor DRM, a un permiso de uso dun determinado libro virtual e até unha data concreta.

A ferramenta de administración da Aula Virtual incorpora a funcionalidade de xerar códigos de activación para un libro virtual que serán válidos até unha data de caducidade determinada, mais a distribución dos códigos é algo que non se fai desde a plataforma, senón que é a propia editora a que se encarga de facerllos chegar aos docentes que utilizan o libro de texto en papel, sendo estes á súa vez os encargados de distribuílos entre o seu alumnado.

Logo, cando un usuario descarga o libro virtual e o abre por primeira vez, váiselle pedir que introduza un código de activación que será validado polo servidor DRM (para o que será preciso dispor de conexión a Internet) para comprobar que o código introducido é correcto, está vixente e refírese ao libro virtual en cuestión. Unha vez validado e permitido o acceso ao libro virtual, o código queda asociado na base de datos de autorizacións ao dispositivo no que se instala (a través do seu *device ID*). Isto permite que un usuario poida volver a activar o libro no mesmo dispositivo empregando o mesmo código de activación, por exemplo no caso de que borre o libro de forma accidental. A figura 7.3 amosa o proceso de descarga e activación dun libro dixital protexido da Aula Virtual de Baía:

1. O usuario accede á Aula Virtual de Baía Edicións a través de Internet, e descarga un libro virtual protexido.

2. Os administradores de Baía Edicións acceden á ferramenta de administración

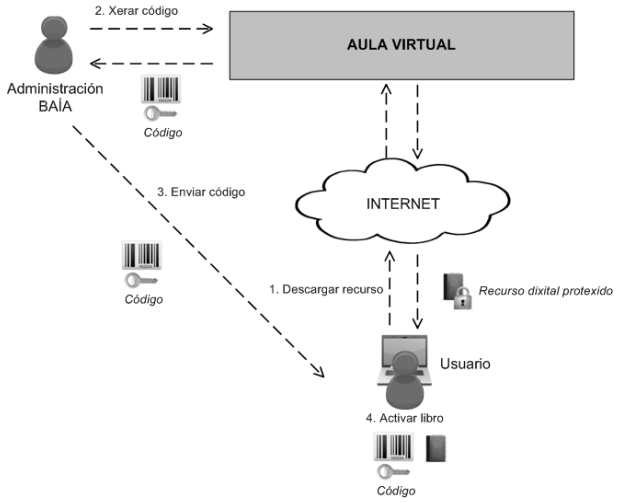

Figura 7.3: Aula Virtual Baía. Control de uso mediante códigos promocionais.

da Aula Virtual e xeran un código de activación asociado ao libro e cunha data de validez limitada temporalmente.

3. O usuario recibe o código de activación que a editora lle envía a través dunha canle independente da plataforma.

4. O usuario abre o libro virtual descargado e introduce o código de activación que, unha vez verificado polo servidor DRM da plataforma, permítelle acceder aos contidos do libro.

Por último, indicar que de cara a ofrecer a maior facilidade de emprego posible, os libros virtuais distribúense xa integrados dentro da aplicación lectora (en versión de escritorio) como un único arquivo executable Java (JAR). Desta forma, os usuarios que descargan o libro virtual non teñen que descargar a maiores ningunha aplicación lectora para abrilo, e só precisarán dispor dun código de activación vixente. O emprego da tecnoloxía Java vén motivado polo feito de que se trata dunha linguaxe multiplataforma que permite que a mesma aplicación funcione en equipos con diferentes sistemas operativos, como Windows, Linux e Mac OS, co único requisito de que teñan instalada a máquina virtual de Java (Java Virtual Machine, JVM).

7.2.3. OQO Plataforma

En 2013, a creativa OQO Editora inicia un proxecto para levar os seus contidos en formato dixital aos centros educativos galegos e tamén do resto de España. O obxectivo era desenvolver unha plataforma web que permitise catalogar e distribuír todos estes contidos en formato dixital aos centros educativos, de forma controlada (con DRM), para o seu emprego por parte dos docentes e do alumnado, tanto durante as clases como desde a casa. O resultado foi OQO Plataforma, un espazo web desde o que os centros educativos poden acceder ao catálogo de contidos dixitais que ofrece a editora e descargar aqueles para os que dispoñan de autorización.

Dentro do catálogo educativo de OQO Editora atopamos contidos en diferentes idiomas (galego, español, inglés, francés etc.) dirixidos fundamentalmente aos niveis de infantil e primaria, entre os que se inclúen libros (a maior parte deles de ilustracións), vídeos (xerados mediante a técnica de animación *stop-motion*) e interactivos (en tecnoloxía Flash). Os formatos que permite distribuír OQO Plataforma son PDF, vídeos MP4 e HTML[14] (o mesmo formato explicado na sección 7.2.1).

Ademais destes formatos cos que OQO xa contaba, nunha segunda fase do proxecto que abordamos en 2014, definiuse un formato de contido propio (OQO Interactivo) que permite empaquetar nun único arquivo unha serie de recursos ao redor dun conto infantil do catálogo da editora. Ademais do libro e a película, este formato inclúe outros recursos que as aplicacións de lectura empregarán noutros espazos predefinidos (teatro, cómic, debuxo e actividades) e que permiten un elevado nivel de interacción ao usuario, con funcionalidades como a gravación de animacións ou debuxos propios, ou a resolución de actividades.

En canto á modalidade de distribución, OQO Plataforma ofrece os contidos só mediante descarga protexida a través dun formato de distribución propio, que utiliza a solución de DRM forte (sección 7.1.1) e de compresión con `WCSA` (sección 7.1.1) para encapsular o contido orixinal (PDF, Flash, HTML ou OQO Interactivo). Para a visualización dos *ebooks* distribuídos neste formato protexido, desenvolvéronse aplicacións de lectura propias (Visor OQO) en versións de escritorio (Windows, Linux e Mac OS X) e para dispositivos móbiles Android.

Unha funcionalidade innovadora que se desenvolveu neste proxecto foi a do consumo en *streaming* dos contidos a través dunha rede local (intranet). O escenario típico para o que se deseñou esta solución é o dunha aula na que tanto o profesor como os alumnos dispoñen de dispositivos individuais, normalmente *netbooks* ou *tablets*, conectados a unha mesma rede local (con ou sen acceso a Internet). Neste contexto, cada vez máis habitual nas aulas de hoxe en día, consideramos que sería interesante

[14] Este formato emprégase tamén para empaquetar os interactivos Flash embebidos nunha páxina HTML.

a posibilidade de que o profesor poida compartir un contido dixital cos alumnos, sen necesidade de entregar a cada un destes unha copia del. Así, desenvolveuse unha aplicación (Visor OQO Profesorado) coas mesmas funcionalidades de lectura que o visor orixinal, pero que incorpora ademais internamente un compoñente (servidor HTTP) a través do cal pode servir en *streaming* os seus contidos a través dunha rede local. Desta forma, incorporamos no Visor OQO un módulo cliente que comproba se dispón de acceso á rede e nese caso envía unha mensaxe de *broadcast* para detectar presenza dalgunha aplicación de profesorado servindo algún contido dentro da súa rede local. De ser así, o Visor OQO Profesorado responde notificando o seu enderezo dentro da rede, quedando establecida a conexión entre eles.

Con respecto aos tipos de autorizacións existentes na plataforma inclúense:

- Data de caducidade: explicada en detalle na sección 7.1.1.
- Impresión: controla a posibilidade de imprimir os contidos en formato PDF desde as aplicacións de lectura.
- Códigos de activación: a mesma solución explicada na sección 7.2.2.
- Número de aplicacións do profesorado: limita o número de aplicacións Visor OQO Profesorado nas que pode abrirse o contido. A comprobación realízase contra o servidor DRM da plataforma no momento de cargar o contido na aplicación. Este permiso foi pensado para controlar o número de aulas nas que se utiliza dito contido.

Como se comentou anteriormente, en 2014 levamos a cabo unha segunda fase do proxecto na que, ademais da incorporación do novo formato de contido OQO Interactivo, abordamos a integración de OQO Plataforma con GaliciaLe[15], a plataforma de préstamo de libros electrónicos para os usuarios das bibliotecas públicas galegas que a Xunta de Galicia puxo en marcha en outubro de 2014, e na que OQO Editora actúa como provedora de contidos dixitais.

GaliciaLe utiliza a solución DRM de Adobe para controlar os préstamos dos *ebooks*, polo que os contidos que distribúe están restrinxidos aos formatos PDF e EPUB, que son os que o sistema ADEPT permite protexer. Isto deixa fóra a moitos dos contidos de OQO, concretamente aos que están en formato HTML, MP4 e OQO Interactivo. A fin de permitir o préstamo deste tipo de contidos distribuídos en OQO Plataforma a través de GaliciaLe, foi preciso abordar unha integración de ambas plataformas, a través de servizos web REST[16], na que OQO Plataforma actúa como provedora externa dos contidos de OQO que GaliciaLe ofrece aos seus usuarios[17].

[15] `http://www.galiciale.gal`
[16] `http://www.restdoc.org/spec.html`
[17] `http://galiciale.oqo.gal`

Para levar a cabo esta integración foi preciso abordar principalmente dous problemas derivados da existencia de diferentes tipos de autorizacións e sistemas de protección dos contidos empregados por cada unha das plataformas. Dunha banda, os contidos de OQO están protexidos mediante o emprego dun formato de distribución propio, e o control sobre o seu uso efectúase nas aplicacións de lectura tamén propias (as diferentes versións de Visor OQO) mediante a comunicación entre estas e o servidor DRM integrado en OQO Plataforma. Isto implica que un usuario que recibe o préstamo dun contido de OQO vai ter que instalar no seu dispositivo de lectura a aplicación Visor OQO para poder acceder a el. Para facer que o préstamo de recursos aloxados por ela mesma ou pola plataforma de OQO sexa o máis transparente posible para os seus usuarios, GaliciaLe incorpora na interface de acceso ao préstamo dun recurso de OQO a ligazón para a descarga das aplicacións de lectura propia, ademais da ligazón para a descarga desde o repositorio de OQO Plataforma do propio recurso protexido.

Doutra banda, o modelo de préstamo no que GaliciaLe ofrece os *ebooks* non ten unha correspondencia directa na plataforma de OQO. Como vimos, si que existía unha autorización data de caducidade que se pode empregar para controlar a data de fin de préstamo, pero outras funcionalidades como a devolución anticipada dun préstamo, a súa renovación ou o control do número máximo de dispositivos no que se pode visualizar un *ebook* non estaban soportadas en OQO Plataforma. Por isto, foi necesario modificar o sistema DRM para incluír as novas autorizacións, o que supuxo desenvolvemento tanto no lado servidor como nas aplicacións de lectura.

Para rematar, descríbese o protocolo definido para a comunicación entre as dúas plataformas (a través de servizos web REST). En primeiro lugar amósanse as operacións que utiliza GaliciaLe para realizar peticións a OQO Plataforma, como a solicitude de información dos recursos dispoñibles para préstamo ou referidas á propia xestión dos préstamos. Logo amósanse tamén as operacións que OQO Plataforma emprega para notificar a GaliciaLe certa información que podería resultarlle de utilidade.

- **Comunicacións GaliciaLe – OQO Plataforma**
  - Obter catálogo de recursos.
    Parámetros: -
    Retorno: a lista de recursos dispoñibles para préstamo.
    Erros: -
  - Obter recurso.
    Parámetros: o identificador do recurso.
    Retorno: a información do recurso.
    Erros: o recurso non existe ou non está dispoñible para préstamo.
  - Prestar recurso.
    Parámetros: o identificador do recurso prestado e a data de caducidade

do préstamo.
Retorno: un identificador do préstamo, que se empregará logo para realizar outras operacións sobre el, como a devolución ou renovación.
Erros: o recurso non existe ou non está dispoñible para préstamo.

- Obter préstamos.
  Parámetros: -
  Retorno: a listaxe dos préstamos realizados.
  Erros: -
- Obter información préstamo.
  Parámetros: o identificador do préstamo.
  Retorno: a información completa do préstamo, que inclúe o recurso prestado, o estado, a data de caducidade e información das instalacións nos diferentes dispositivos do usuario.
  Erros: o préstamo non existe.
- Obter ligazón descarga.
  Parámetros: o identificador do préstamo.
  Retorno: a *URL* de descarga do recurso protexido.
  Erros: o préstamo non existe ou non se atopa aínda dispoñible para descarga (durante o empaquetado do recurso no formato de distribución propio).
- Renovar préstamo.
  Parámetros: o identificador do préstamo e a nova data de caducidade.
  Retorno: a confirmación da renovación.
  Erros: o préstamo non existe ou data de caducidade está xa superada.
- Devolver préstamo.
  Parámetros: o identificador do préstamo.
  Retorno: a confirmación da devolución.
  Erros: o préstamo non existe ou ten aínda algunha instalación activa en algún dispositivo do usuario, polo que non é posible efectuar a devolución.

- **Comunicacións OQO Plataforma – GaliciaLe**
  - Notificar devolución préstamo: cando un usuario devolve o recurso en todos os visores nos que o ten instalado, OQO Plataforma notifica a GaliciaLe para que, se o considera oportuno, poida marcar o préstamo como devolto.
    Parámetros: o identificador do préstamo.
    Retorno: -
    Erros: -
  - Notificar actualización recurso: cando se actualiza en OQO Plataforma a información descritiva dun recurso que está dispoñible para préstamo, notifícase a GaliciaLe para que esta poida solicitar a nova información e

actualizala na súa base de datos.
Parámetros: o identificador do recurso.
Retorno: -
Erros: -

7.2.4. Viateca

Viateca (`http://viainteligente.cli.enxenio.net/Servicios/Viateca`) é un dos servizos que desenvolvemos en Enxenio en 2013 para a Vía Inteligente[18], un proxecto orientado á definición dun entorno metodolóxico e tecnolóxico para a creación dunha nube de servizos orientados á cidadanía e ofrecidos a través de redes sen fíos instaladas no pavimento das cidades. Estes servizos foron especialmente deseñados para o seu funcionamento en dispositivos móbiles e tendo en conta a limitación de ancho de banda de 256Kbps que a *Comisión Nacional de los Mercados y la Competencia* fixa para a oferta de servizos gratuítos a través de redes wifi. Concretamente, trátase dunha plataforma de distribución de *ebooks* en *streaming*, que integra a solución para a distribución comprimida dos textos explicada na sección 7.1.2.

Figura 7.4: Viateca. Catálogo, ficha dun libro e vista de lectura.

[18]Proxecto ENVÍA financiado polo Fondo Europeo de Desarrollo Regional (FEDER) e cofinanciado polo Ministerio de Industria, Energía y Turismo dentro dol Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica 2008-2011 (TSI-020302-2011-6).

Esta plataforma dispón dunha interface web sinxela e moi intuitiva que permite aos usuarios navegar polo catálogo de *ebooks* dispoñibles para a lectura, filtralos por xénero ou idioma, e buscar por título. Ao premer sobre a portada dun libro amósase unha ficha de catalogación, que inclúe información como o título, a autoría, a edición, o xénero e a sinopse, ademais de ofrecer acceso á lectura do mesmo. Para a lectura, integra un visor deseñado especialmente para ofrecer unha correcta visualización dos contidos en dispositivos móbiles con capacidades de procesamento e tamaño de pantalla moi diversos, algo que resulta de vital importancia nos servizos da *Vía Inteligente*. Este visor inclúe as funcionalidades máis básicas das aplicacións de lectura de libros electrónicos, como a navegación secuencial e a través do índice de contidos, ou o axuste do tamaño da fonte para a lectura. En realidade, trátase dunha primeira versión moi preliminar de Xcloud Reader (sección 7.1.2), e que incorpora tamén a funcionalidade de lectura *offline* para aqueles navegador con soporte das funcionalidades *offline web applications* e *local storage* de HTML5. A figura 7.4 amosa as interfaces do catálogo, da ficha dun libro e do lector.

7.2.5. Xcloud Bookstore

A principios de 2014, presentamos en Enxenio Xcloud Bookstore (`http://www.xcloud-bookstore.com`), unha tenda de libros electrónicos en formato EPUB para lectura en *streaming* no visor Xcloud Reader (sección 7.1.2) integrado.

Trátase dun produto dirixido principalmente a editoras que queiran dispor dunha plataforma web propia e con deseño personalizado para a comercialización dos seus libros electrónicos, e polo tanto, sen ter que pagar as comisións (normalmente o 30 % dos ingresos por venda) que aplican as plataformas que actúan de intermediarias como as de Amazon ou Apple. Xcloud Bookstore ofrécese como un paquete estándar, que inclúe todas as funcionalidades básicas (xestión do catálogo, clientes, vendas a través de PayPal[19], Xcloud Reader integrado para lectura etc.), e distintos módulos funcionais e servizos adicionais (posibilidade de venda por subscrición, integración de TPV virtual específico, servizo de *hosting* e *backups*, configuración avanzada de estilos etc.) que permiten a cada editora axustar a plataforma ás súas necesidades.

Funcionalmente divídese en dúas seccións diferenciadas que son, a *sección pública*, desde a que os usuarios poden rexistrarse, navegar polo catálogo de *ebooks*, mercalos e acceder á súa lectura; e doutra banda está a *ferramenta de administración*, con acceso restrinxido, desde a que é posible xestionar o catálogo de libros e outra información de xestión (contactos, vendas, estatísticas de uso, lanzar campañas de promoción dos libros etc.).

Sección pública. É a parte da aplicación de acceso público, e que ofrece as funcionalidades de navegación polo catálogo, compra e lectura dos *ebooks*. Dispón

[19] `https://www.paypal.com`

de dúas interfaces alternativas, unha estándar e outra optimizada para a súa visualización desde dispositivos móbiles, ofrecendo así unha moi boa usabilidade en todo tipo de dispositivos actuais.

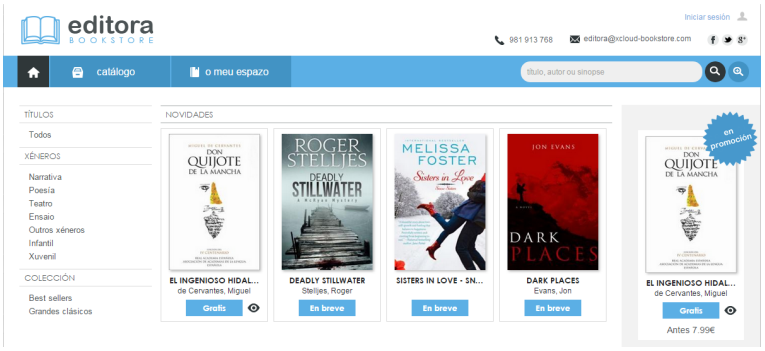

Figura 7.5: Xcloud Bookstore. Páxina de inicio.

A páxina de inicio (figura 7.5) amosa tres seccións, unha coas últimas novidades incorporadas, outra cos títulos máis destacados da editora, e outra cos máis vendidos. En canto ás funcionalidades de navegación, Xcloud Bookstore permite ver o catálogo de *ebooks* dispoñibles organizados por título, xénero, colección editorial ou por autor. Ademais da navegación polo catálogo, a plataforma tamén ofrece a posibilidade de atopar un libro concreto de forma sinxela a través dunha ferramenta de procura que dispón de dúas interfaces diferentes: o buscador simple e o buscador avanzado. O buscador simple realiza unha busca textual no título, autoría e sinopse dos *ebooks*, mentres que o buscador avanzado permite especificar os campos nos que se quere buscar (busca por *metadatos*).

De cada libro amósase en primeira instancia unha ficha resumo que inclúe a súa capa, o título, a autoría, o prezo e unha ligazón para acceder á mostra gratuíta do *ebook*, de estar dispoñible. Premendo enriba da capa ou do título é posible acceder á ficha completa (figura 7.6) con información adicional como a sinopse, edición, idioma, número de páxinas (da edición en papel), así como unha listaxe de títulos relacionados que Xcloud Bookstore obtén de forma automática.

Para ter acceso ás funcionalidades de compra e lectura un usuario ten que rexistrarse na tenda a través do correspondente formulario rexistro, no que terá que introducir o seu nome e apelidos, un enderezo de correo electrónico, o contrasinal de acceso e, opcionalmente, un alcume e a data de nacemento. Tamén será preciso que acepte de forma explícita as condicións de uso da plataforma. Unha vez cumprimentado o formulario de rexistro, o usuario recibirá un correo electrónico cunha ligazón de activación, que terá que seguir para completar o proceso de rexistro. A partir de ese intre, o usuario estará activo na plataforma e poderá acceder ao seu

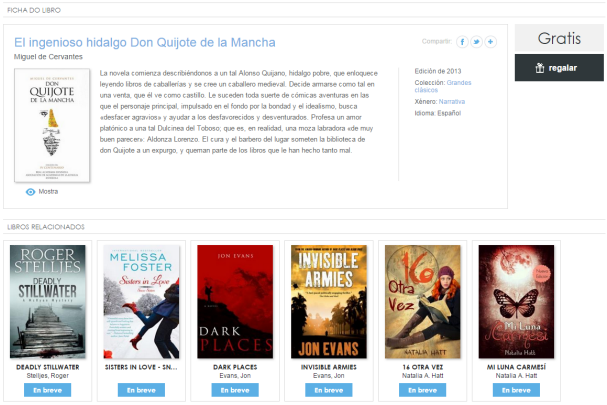

Figura 7.6: Xcloud Bookstore. Ficha completa dun *ebook*.

espazo persoal introducindo o seu correo electrónico e contrasinal.

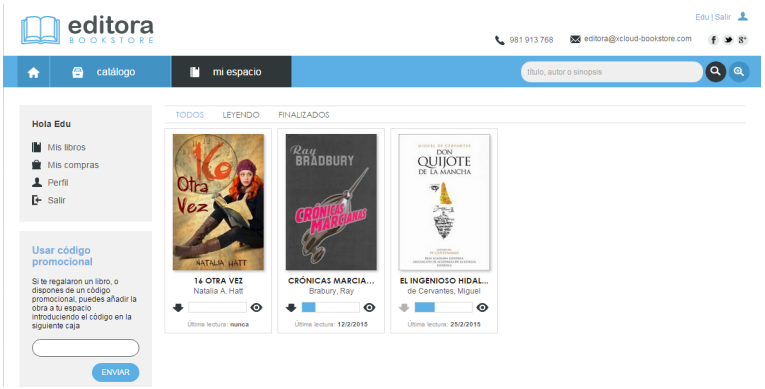

Figura 7.7: Xcloud Bookstore. Espazo persoal.

Dentro do espazo persoal (figura 7.7), o usuario vai ter acceso aos libros que ten mercados organizados segundo o seu estado de lectura (todos, lendo actualmente e rematados). Para cada libro amósase a capa, o título, unha barra indicativa do progreso e a data da última lectura, un botón para abrir o libro (no visor Xcloud Reader), e un botón para a funcionalidade de descarga manual que permite facer dispoñible o libro para a lectura sen conexión. Esta funcionalidade, xunto co resto de características da aplicación lectora foron explicadas en detalle na sección 7.1.2. A figura 7.8 amosa as interfaces que ofrece a versión de Xcloud Reader integrada en Xcloud Bookstore para proporcionar acceso ás diferentes funcionalidades de usuario que ofrece.

Figura 7.8: Xcloud Bookstore. Interfaces Xcloud Reader (navegación, índice, ficha descritiva e preferencias de lectura).

Xcloud Bookstore permite integrar diferentes sistemas de pago electrónico, aínda que na súa instalación básica incorpora PayPal, que permite o pago a través de contas propias de PayPal mais tamén con tarxeta de crédito e débito. O proceso de compra é moi sinxelo para o usuario, que ao premer sobre o prezo do libro accede directamente ao formulario de pago seguro de PayPal para finalizar o mesmo. Unha vez procesado o pago, e confirmada a súa validez, o usuario disporá de forma inmediata do *ebook* no seu espazo persoal.

Cando se realiza unha compra na plataforma, xérase un recibo dixital que o usuario poderá ver dentro da sección de "As miñas compras" no seu espazo persoal. Se ademais precisa que se lle emita unha factura, terá a posibilidade de solicitala no momento da compra. Nese caso, o sistema pediralle que introduza os datos de

facturación (enderezo, NIF etc.) e xerará unha factura en PDF que o usuario poderá descargar.

Por último, indicar que Xcloud Bookstore ofrece a posibilidade de mercar un *ebook* para regalo. Nese caso, o beneficiario recibirá un correo electrónico coa notificación e un código promocional que poderá introducir no seu espazo persoal e obter así acceso ao libro.

Ferramenta de administración. É a parte da plataforma de acceso restrinxido, desde a que a editora vai poder realizar a xestión do seu catálogo de *ebooks* e toda a demais información de vendas e uso da plataforma (figura 7.9). As seccións que se inclúen dentro da ferramenta de administración son:

Figura 7.9: Xcloud Bookstore. Interface administración.

- Libros: permite xestionar o catálogo de *ebooks* dispoñibles para a venda. Para a introdución dun novo libro, será preciso cubrir a súa información descritiva e de catalogación (título, autoría, sinopse, xénero, ISBN etc.) e subir o arquivo do *ebook* en formato EPUB. Opcionalmente é posible subir unha mostra do libro, tamén en formato EPUB, que estará dispoñible de forma gratuíta para todos os usuarios da plataforma (rexistrados ou non).

- Autores/as: permite xestionar a información dos autores dos libros subidos á plataforma.

- Coleccións: permite a creación e xestión de coleccións editoriais para a organización dos *ebooks*.

- Promocións: permite a realización de campañas de promoción dos *ebooks*, baseadas na distribución gratuíta e controlada, a través dun sistema de códigos promocionais, de exemplares dixitais dos libros para a súa lectura desde a

propia plataforma. A distribución dos códigos promocionais pode facerse desde a propia plataforma ou á marxe dela:

  - Desde Xcloud Bookstore: a plataforma distribúe automaticamente os códigos promocionais mediante o envío de correos electrónicos aos contactos dunha categoría. A creación de contactos e xestión das categorías realízase desde a seguinte opción da ferramenta de administración (Contactos).
  - Fóra de Xcloud Bookstore: a plataforma permite xerar un número determinado de códigos promocionais que se exportan como un arquivo de texto, sendo responsabilidade da editora a posterior distribución deses códigos por calquera outra canle.

- Contactos: permite a creación de contactos (nome, enderezo de correo electrónico e unha referencia descritiva) e a súa organización por categorías para o seu emprego posterior como destinatarios das promocións de *ebooks* controladas pola plataforma.

- Estatísticas: ofrece distinta información estatística do uso da plataforma, organizada en información de usuarios e de libros. Para cada usuario amósase, ademais dos seus datos de rexistro (nome e apelidos, enderezo de correo electrónico, data de rexistro) información estatística de uso da plataforma como a data de último acceso, o número total de accesos, o número de libros mercados, o número de libros dispoñibles no seu espazo persoal (incluídos os regalados), o número de accesos ao lector, e a data de última lectura. Para un libro amósase o seu título, a data de subida, o número de vendas, o beneficio obtido polas vendas, o número de exemplares dispoñibles nos espazos dos usuarios (incluídos os regalados mediante promocións), o número de accesos para lectura, o número de usuarios que teñen o libro no seu espazo pero que aínda non comezaron a lelo, o número de usuarios que o están lendo e o número de usuarios que remataron de lelo.

- Vendas: ofrece información relacionada coas vendas de *ebooks* efectuadas na plataforma, incluídos os datos de facturación e a copia dixital da factura, naquelas vendas nas que o cliente a solicitou no momento da compra.

- Idiomas: permite xestionar os idiomas soportados para a publicación de libros na plataforma.

Para rematar, indicar que Xcloud Bookstore obtivo a certificación internacional de calidade de produtos software ISO/IEC 25000[20], o que acredita que o seu desenvolvemento é conforme á especificación de requisitos e a superación con éxito

[20] `http://iso25000.com`

do proceso de avaliación dunha serie de características de calidade do software que permiten garantir o seu mantemento futuro.

Capítulo 8

Creación de contidos

Non sería de todo certo afirmar que as pequenas editoras non dispoñen de ferramentas para a creación de *ebooks*. En primeiro lugar, porque os programas típicos que empregan para a edición dos libros en papel, como Adobe InDesign[1] ou QuarkXPress[2] ofrecen nas súas versións máis recentes a posibilidade de exportar directamente os contidos a formatos de publicación dixital como PDF, EPUB ou MOBI. Ademais, existen multitude de aplicacións *standalone* e web para a edición de libros electrónicos dispoñibles en Internet, tanto de pago como incluso gratuítas.

Unha característica común á maioría destas aplicacións é que condicionan o proceso de edición ao formato de publicación do *ebook* e, incluso, ao seu aspecto gráfico. Desta forma, o habitual é que cada aplicación permita xerar un tipo moi particular de *ebook*, co cal para crear un tipo concreto de publicación hai que atopar unha ferramenta axeitada e aprender o seu manexo, e de querer editar un mesmo libro en distintos formatos ou con diferentes acabados gráficos, será preciso empregar distintas ferramentas e repetir todo o proceso de edición en cada unha delas. Mesmo sendo posible na maioría dos casos reutilizar (copiando e pegando) os textos e demais recursos dixitais introducidos nunha ferramenta en outra, resulta obvio que manter os contidos por duplicado en distintas ferramentas complica o proceso de creación e, sobre todo, de revisión editorial.

Existen tres alternativas básicas para a creación de libros electrónicos e, dependendo da alternativa escollida, diferentes ferramentas que a soportan:

- Exportación desde programas de edición: a primeira solución que atopan as editoras para a creación dos seus títulos en formato *ebook* é a través

[1] `http://www.adobe.com/es/products/indesign.html`
[2] `http://www.quark.com/Products/QuarkXPress`

das funcionalidades específicas que incorporan nas súas últimas versións os programas clásicos para a edición das maquetas dos libros en papel. Nun primeiro intento por ofrecer unha solución unificada para a edición conxunta do libro en papel e dixital, programas como InDesign ou QuarkXPress incorporaron a posibilidade de exportar como libro electrónico os contidos do libro editados para impresión. A pouco que o libro tivese algún elemento gráfico ou estético, e non fose só texto corrido, o resultado acadado na exportación era moi pobre, e os *ebooks* resultantes non alcanzaban unha calidade suficiente como para ser comercializados. Mesmo non se tiña garantía de que os *ebooks* exportados resultasen conformes á especificación estándar dos formatos, e moitas veces fallaba o proceso de validación requirido para a publicación dos *ebooks* en algunhas plataformas de terceiros. Non obstante, isto non impediu que moitas editoras distribuísen igualmente estas versións provocando a insatisfacción dos seus clientes, en parte por un intento desesperado de entrar no negocio dixital e en parte por nin sequera ser conscientes dos problemas de visualización que o *ebook* ofrecería aos lectores.

Co tempo, os programas clásicos de edición foron mellorando as súas funcionalidades de exportación a formato *ebook*, mais existe un aspecto básico que, máis alá do soporte, diferencia unha edición en papel dun libro dunha dixital, e que estes programas non van poder obviar. Trátase do feito de que unha edición en formato de impresión parte dun tamaño físico fixo ao que é preciso axustar os elementos de contido, mentres que nunha versión dixital moderna (nun formato fluído) os contidos teñen que adaptarse a distintos tamaños e resolucións de pantalla, o que condiciona de forma decisiva a súa organización. Así, por exemplo, se ben nunha publicación en papel é posible xogar coa colocación dos elementos de contido nas páxinas, nunha edición dixital é imprescindible establecer unha ordenación lineal para permitir que as aplicacións de lectura os amosen de forma ordenada aos lectores, con independencia das características físicas do dispositivo de lectura. Isto fai que as decisións a tomar durante o proceso de edición sexan diferentes de tratarse dun libro físico ou dun *ebook*, sendo esta a causa principal de que a exportación directa en formato dixital dun libro editado para impresión non ofreza un resultado de calidade.

Como consecuencia disto, os programas de edición van incorporando certas funcionalidades que permiten ao editor do libro especificar como debe interpretarse cada elemento de contido na exportación a formato dixital (definir estilos mediante regras CSS, establecer orden de lectura, colocación das imaxes en liña co texto, eliminación de espazos forzados etc.), o que acaba por complicar moito o proceso de edición ao que está acostumado para obter como resultado un *ebook* que, as máis das veces, non cumpre coas súas expectativas.

- Conversión de formato: outra alternativa para a edición dun *ebook* nun formato

dixital concreto é a conversión desde outro formato dixital[3]. O caso máis típico é a conversión dun libro en formato PDF a un formato fluído como EPUB ou MOBI. En realidade esta é, ou debería selo, máis unha alternativa para que os usuarios poidan transformar os seus propios *ebooks* e documentos nun formato máis accesible, que unha solución para as editoras, xa que o resultado que se obtén na conversión é normalmente bastante deficiente.

Un exemplo de ferramenta que, entre moitas outras funcionalidades, permite a conversión de *ebooks* desde una gran cantidade de formatos de entrada (EPUB, HTML, PDF, RTF, txt, cbc, fb2, lit, MOBI, ODT, prc, pdb, PML, RB, cbz e cbr) a distintos formatos de saída (EPUB, fb2, OEB, lit, lrf, MOBI, pdb, pml e rb) é Calibre[4].

- Edición dixital: esta alternativa consiste en abordar o proceso de creación dun libro dixital desde cero, empregando algunha ferramenta deseñada especificamente para a creación de *ebooks*, e tomando, xa desde o principio do proceso creativo, as decisións axeitadas que faciliten a súa lectura desde os distintos tipos de dispositivos. Esta é, sen dúbida, a mellor das alternativas para a edición de *ebooks*, e a que permite obter os mellores resultados en canto á calidade dos libros producidos, e consecuentemente, a satisfacción dos lectores.

Existen multitude de ferramentas para a edición dixital de *ebooks*, tanto aplicacións de escritorio como web, sendo o habitual que ditas ferramentas foran especialmente deseñadas para a creación dun tipo moi específico de contido e a súa exportación nun formato publicación tamén moi concreto.

Para a edición dixital de creacións literarias constituídas fundamentalmente por texto con algunha imaxe incorporada, unha das ferramentas máis coñecidas é Sigil[5], que permite exportar os *ebooks* en formato EPUB. Trátase dunha aplicación de escritorio *open source* multiplataforma (Windows, Mac e Linux), que ten como punto forte a facilidade de uso a través dunha interface moi semellante á dun editor de texto. En cambio, non permite a introdución de elementos como vídeos, sons ou contido interactivo, polo que o seu emprego está practicamente restrinxido a *ebooks* de formato sinxelo. En febreiro de 2014 Sigil deixa de ser un proxecto activo, e os seus propios creadores recomendan o emprego de Calibre, outra aplicación de escritorio xa comentada anteriormente como alternativa para a conversión de formato dos libros electrónicos, que incorpora tamén un editor de *ebooks* e permite exportar aos formatos EPUB e azw3 (Kindle). O editor de *ebooks* de Calibre ten unha interface amigable e intuitiva, moi semellante á de Sigil, pero presenta tamén as mesmas limitacións en canto ao tipo de contidos que se poden incorporar aos *ebooks*. En

[3] En sentido estrito, non sería un proceso de creación dun *ebook* ao partir xa do libro dixitalizado.
[4] `http://calibre-ebook.com`
[5] `http://sigil-ebook.com`

canto ás alternativas web para a edición de *ebooks* a través de Internet, na maior parte dos casos trátase de plataformas que ofrecen diferentes servizos editoriais e que integran entre as súas funcionalidades algunha relacionada coa edición de contidos. Unha destas plataformas é Byeink[6], que ofrece aos seus usuarios un sinxelo editor para a creación e introdución dos contidos dos *ebooks* que posteriormente permite vender directamente desde ela.

Ademais do sector literario, outra das áreas con maior demanda de contidos dixitais na actualidade é o sector da educación, o que propiciou a aparición de diferentes ferramentas que permiten a creación de contidos destinados ao seu emprego nas aulas. A diferenza das creacións literarias constituídas tipicamente por texto e algunha imaxe, os *ebooks* educativos incorporan elementos multimedia e interactivos, que proporcionan unha experiencia de aprendizaxe enriquecida e ofrecen un valor engadido ao emprego do libro de texto en papel. Unha mostra do gran auxe que están a experimentar as tecnoloxías educativas é o feito de que algunhas das grandes tecnolóxicas como son Google con Course Builder[7], Apple con iBooks Author[8] ou Microsoft con LCDS[9] ofrezan as súas propias solucións para a creación e o emprego de contidos dixitais educativos. Outras aplicacións, neste caso de escritorio, moi empregadas para a creación de contidos dixitais educativos son eXeLearning[10] ou HotPotatoes[11].

8.1. E-ditor

Nesta sección preséntase E-ditor (`http://www.e-ditor.es`), unha ferramenta na nube para a edición de *ebooks* que desenvolvemos co obxectivo de poñer a disposición de editoras e outras organizacións creadoras de contidos unha solución para a creación e mantemento do seu catálogo completo de contidos dixitais.

E-ditor dispón de dúas modalidades de uso: unha *persoal* (gratuíta) dirixida a aqueles usuarios que queiran elaborar os seus propios contidos para uso persoal (limitadas a 50MB de espazo); e outra *profesional* (de pago) dirixida a empresas do sector editorial, a motivación orixinal do proxecto, que precisan comercializar os *ebooks* creados coa ferramenta, e que van requirir un maior espazo de almacenamento, rexistrar usuarios colaboradores, personalizar os contidos xerados ou un servizo de soporte técnico. Na actualidade hai rexistrados cerca de 900 usuarios, e a previsión é que serán máis de 1000 a principios de 2016, cando se cumpren tres anos da estrea

[6] `https://byeink.com`
[7] `https://code.google.com/p/course-builder/`
[8] `http://www.apple.com/es/ibooks-author/`
[9] `https://www.microsoft.com/en-us/learning/lcds-tool.aspx`
[10] `http://exelearning.net/`
[11] `https://hotpot.uvic.ca/`

de E-ditor en 2013. Deles, a maioría son usuarios da modalidade persoal, mais tamén hai varias editoras que están empregando a ferramenta na modalidade profesional.

E-ditor foi desenvolvido como un proxecto experimental cun alto grado de innovación, cuns obxectivos ambiciosos que, finalmente, consideramos cumpridos. Un aspecto moi importante para levar a cabo un proxecto destas características foi a posibilidade de contar durante a definición da ferramenta coa colaboración de diferentes empresas do sector editorial, o que nos permitiu coñecer de primeira man as dificultades coas que se atopaban estas para a edición dixital dos seus contidos e, posteriormente, validar as distintas funcionalidades desenvolvidas.

Os principais elementos diferenciais que presenta E-ditor con respecto a outras ferramentas de edición de *ebooks* é a súa orientación cara unha "representación semántica" dos contidos e a separación do proceso creativo en fases. A hipótese da que partimos foi a de que no momento da creación non debería ser preciso ter en conta aspectos relacionados cos estilos de presentación nin co formato final de publicación, senón centrarse só nos contidos en si mesmos e na súa organización. E viceversa, debería ser posible traballar nos estilos de presentación dunha publicación sen necesidade de saber a priori os contidos concretos que vai ter, ou incorporar un novo formato de publicación sen que iso implique ter que adaptar os contidos existentes a el.

Desta idea sinxela derívanse unha serie de implicacións que resultan moi interesantes. Por exemplo, a separación das fases de elaboración dos contidos e de exportación vai permitir que un mesmo contido, introducido unha única vez, poida ser exportado a distintos formatos (EPUB, HTML, SCORM, *app* Android etc.) e con diferentes aspectos a nivel gráfico.

A continuación amósanse cales son as principais características de E-ditor, destacando tamén aquelas que fan desta ferramenta unha alternativa moi útil para que as empresas do sector editorial manteñan o seu catálogo de publicacións:

- Centrado nos contidos: sen dúbida, o máis importante dunha publicación son os seus contidos. E-ditor representa e almacena os contidos das publicacións de forma estruturada/semántica nunha base de datos, o que lle permite acadar as seguintes vantaxes:
  - O contido é independente do formato de publicación final, o que permite que un mesmo contido poida ser exportado en diferentes formatos de publicación.
  - Simplifica o proceso de actualización e a revisión para corrección de erros dos contidos durante o ciclo de vida das publicacións. Estas actualizacións e correccións efectúanse directamente sobre o contido e proxéctanse automaticamente a todos os acabados e formatos de publicación dispoñibles para o *ebook*.

  - Permite a reutilización de contidos entre publicacións, ao estaren estes representados de acordo coa súa natureza e non con ningunha especificación final de formato.
  - Favorece o mantemento dos contidos no tempo, garantindo a capacidade da editora de adaptar o seu catálogo completo de *ebooks* ás evolucións tecnolóxicas futuras de forma sinxela e económica, sen ter que abordar un proceso de reedición das súas publicacións.

- Aplicación SaaS (*Software as a Service*): permite traballar directamente cos contidos aloxados no servidor a través de Internet. Algunhas das vantaxes que ofrece isto son:
  - Accesibilidade dos contidos, xa que se pode acceder a eles desde calquera dispositivo con conexión a Internet e sen que sexa preciso instalar ningún software adicional (desde o navegador web).
  - Facilidade para a xestión dos contidos, ao traballar sempre coa versión actualizada dos mesmos.
  - Actualizacións e novas funcionalidades dispoñibles de forma inmediata para os usuario.
  - Alta dispoñibilidade e seguridade dos contidos (mediante *backups* diarios dos contidos).
- Repositorio centralizado: mediante o emprego dunha única ferramenta, unha editora vai poder manter o seu catálogo completo de publicacións dixitais.
- Soporte ao proceso de creación: a través da definición dun fluxo de traballo (*workflow*) que separa o proceso creativo en distintas fases (definición da estrutura, introdución dos contidos, deseño gráfico etc.).
- Edición colaborativa e especializada: ao permitir o acceso ao catálogo compartido de publicacións a diferentes usuarios que, ademais, poden ocuparse das tarefas concretas nas que están especializados dentro do proceso de creación dos *ebooks*. Así, E-ditor permite que as organizacións que dispoñen de persoal con distintas responsabilidades no proceso editorial poidan organizar mellor o seu traballo, sen que este interfira co dos demais, e favorecer así unha mellora na produtividade.
- Soporte para publicacións "seriadas": resulta habitual que as editoras dispoñan no seu catálogo de coleccións editoriais formadas por unha serie de títulos que comparten unha mesma estrutura organizativa dos contidos, un mesmo deseño gráfico e un mesmo formato de publicación. E-ditor ofrece soporte a este tipo de publicacións grazas á separación en fases do proceso de creación, o que permite definir unha estrutura e un estilo común para as publicacións da colección, e crear cada un dos seus números simplemente introducindo os seus contidos concretos.

8.1.1. Arquitectura

A arquitectura do sistema desenvolvido segue o modelo tradicional das aplicacións cliente-servidor a través de Internet, onde E-ditor actúa do lado servidor ofrecendo un acceso centralizado aos servizos de creación dos *ebooks* e os usuarios, do lado de cliente, acceden a ditos servizos a través do navegador web dos seus dispositivos.

Centrándonos na parte de creación e desde o punto de vista da arquitectura funcional, a ferramenta divídese en catro compoñentes básicos: o Módulo de Rexistro/Autenticación, o Módulo de Edición, o Módulo de Presentación e o Módulo de Exportación. Esta separación lóxica en módulos permite dar soporte ás diferentes etapas do proceso de creación de contidos dixitais.

Módulo de Rexistro/Autenticación

Este módulo é o encargado de controlar o rexistro dos usuarios na ferramenta, e o seu acceso a través dun sistema de identificación baseado no emprego dun enderezo de correo electrónico e un contrasinal. Ademais, E-ditor incorpora a posibilidade de que un usuario rexistre como colaboradores a outros usuarios, dándolle desta forma acceso ao seu catálogo de publicacións. A xestión e o control de acceso dos usuarios colaboradores tamén é responsabilidade deste módulo.

Módulo de Edición

Este módulo é o responsable de ofrecer as funcionalidades de creación das publicacións, a súa estrutura e os contidos que inclúe. Á súa vez, divídese en tres submódulos funcionais que son:

- Catálogo: permite xestionar os *metadatos* da publicación, segundo a especificación DCMES (*Dublin Core Metadata Element Set*).
- Definición da Estrutura: permite establecer a estrutura xerárquica na que se organiza a publicación (capítulos, seccións, subcapítulos etc.), e os tipos de contidos (texto, exercicios, glosario etc.) que se permiten en cada un dos seus niveis. Convén matizar aquí a diferenza existente entre a estrutura e o índice de contidos propio da publicación, que se crea a través das funcionalidades que ofrece o submódulo Edición de Contidos. A estrutura é á organización de alto nivel da publicación, isto é, unha especie de modelo para a posterior creación do que serán os contidos da mesma. Se facemos unha analoxía coa terminoloxía empregada na programación orientada a obxectos, os elementos da estrutura serían o equivalente ás clases mentres que os elementos de contido serían o equivalente aos obxectos que se crean como instancias das primeiras.

O obxectivo de definir a estrutura da publicación é prefixar o modo no que se van organizar os seus contidos, e permitir que a ferramenta poida controlar e garantir que os contidos introducidos na fase de edición sexan consistentes con dita estrutura. Por un lado, isto vai resultar útil para manter a coherencia tanto ao longo dunha publicación concreta, mesmo cando traballen persoas diferentes en distintas partes dela, como para a elaboración de publicacións seriadas. E doutra banda, establece unha guía para o proceso de edición dos contidos que permite que a ferramenta poña a disposición dos usuarios na fase de edición só aquelas ferramentas de creación permitidas segundo a definición da estrutura.

- Edición de Contidos: ofrece as funcionalidades que permiten a introdución e xestión dos contidos da publicación. Permite a creación do índice de contidos da publicación, que sería a definición dos capítulos, subcapítulos, temas, apartados etc. (segundo a estrutura definida previamente) concretos que inclúe, e tamén a propia edición dos contidos (texto, imaxes, vídeos, definicións, exercicios etc.) incluídos dentro deles. Na actualidade E-ditor inclúe catro tipos de contidos diferentes, que permiten crear practicamente calquera tipo de publicación, e que son:

  - Contido conceptual. É o tipo de contido máis habitual nas publicacións dixitais e está formado basicamente por texto, mais pode incluír outros recursos multimedia como imaxes ou vídeos. En ocasións tamén nos referiremos a el como simplemente texto ou contido HTML, por ser a linguaxe na que se representa internamente.

    Para permitir a súa edición, a ferramenta incorpora un editor WYSIWYM (*What You See Is What You Mean*), que permite a introdución dos contidos e a súa anotación en base a súa natureza semántica, sen ter en conta cal vai ser finalmente o seu formato de representación, o que establece a diferenza cos editores de HTML máis típicos que seguen unha aproximación WYSIWYG (*What You See Is What You Get*). Desta forma, desde este editor vai ser posible crear elementos como notas, destaques, definicións, referencias etc., mais non definir como eses elementos se visualizarán finalmente. Darlle o aspecto final a cada un destes elementos será responsabilidade do Módulo de Presentación que se explicará máis adiante.

  - Exercicios. Permite crear bloques de exercicios dos tipos máis habituais que se poden atopar en libros de texto educativos ou secuencias didácticas, como son preguntas de tipo test, verdadeiro/falso, completar, emparellar etc. A maioría destes exercicios inclúen a posibilidade de introducir a súa solución no momento de crealos, o que permite exportalos como exercicios interactivos que se corrixen automaticamente cando o usuario os realiza, ou dentro de contidos *e-learning* que permiten a avaliación dos usuarios a

través de plataformas de teleformación. Os tipos de exercicios dispoñibles na ferramenta son:

- Test: unha pregunta con varias respostas, das que só unha será a correcta. Permite incluír de forma opcional distintas mensaxes de reforzo (unha para cada resposta) que se van poder empregar para ofrecer unha explicación ao usuario de por que a súa resposta é correcta ou incorrecta.
- Verdadeiro ou falso: un enunciado que pode ser verdadeiro ou falso. No caso de que o enunciado sexa falso é posible incluír, opcionalmente, cal sería o enunciado verdadeiro.
- Resposta curta: unha cuestión que ten unha solución moi concreta e concisa. Permítese a introdución de diferentes variantes da solución, que deberían ser distintas formas de escribir a mesma resposta, co obxectivo de permitir realizar unha avaliación automática da resposta introducida en modo texto polo usuario. Tamén se permite indicar no momento da súa creación se a comprobación da resposta do usuario ten que coincidir de forma exacta con unha das solucións ou se permiten aproximacións, ignorando maiúsculas/minúsculas e acentos.
- Completar: exercicio que consiste nun enunciado ao que lle faltan certas palabras que haberá que completar en base ao seu contexto. Igual que no caso anterior, é posible indicar na súa creación se a comprobación da resposta ten que ser exacta ou pode ser aproximada.
- Emparellar: este exercicio define un conxunto de elementos relacionados dous a dous (parellas) segundo un criterio definido no seu enunciado. O obxectivo é atopar esas parellas de elementos, partindo dunha configuración inicial aleatoria dos mesmos.
- Clasificar: este exercicio define unha serie de categorías xunto con un conxunto de elementos pertencentes a cada unha delas. O obxectivo é determinar os elementos que pertencen a cada categoría partindo dunha configuración inicial aleatoria (pode considerarse unha variante do exercicio de emparellar, onde a relación entre categorías e elementos é unha a varias).
- Ordenar: exercicio consistente en establecer a orde dunha serie de elementos inicialmente desordenados, segundo un criterio definido no seu enunciado.
- Localizar: exercicio formado por un enunciado, que define un criterio, e un texto no que se inclúen algúns elementos (letras, monemas, palabras, frases etc.) que cumpren dito criterio, sendo o obxectivo do exercicio a localización deses elementos.
- Sopa de letras: este exercicio define unha serie de termos segundo un criterio definido no seu enunciado, que se distribúen en posicións

aleatorias dunha matriz ocupando cada letra unha cela, e enchendo o resto das celas con letras aleatorias. O obxectivo é localizar os termos dentro da matriz.

- Redacción: un enunciado que non ten unha solución concreta e concisa, senón que require dunha resposta textual máis ou menos ampla e unha avaliación subxectiva, sendo este o único tipo de exercicio que inclúe a ferramenta que non permite a resolución automática.

- Enlaces. Permite crear seccións de enlaces (*hyperlinks*) tanto a outras partes do contido como externos.
- Glosario. Permite crear unha listaxe de pares *termo-definición*, que habitualmente van ser palabras co seu significado, mais que tamén se pode empregar para representar outra información como, por exemplo, un índice onomástico ou toponímico.

Módulo de Presentación

A responsabilidade deste módulo é a transformación de cada tipo de contido (texto, exercicio, enlaces ou glosario) e cada elemento constituínte (imaxe, vídeo, nota, pregunta test, enlace etc.) no seu formato de presentación final, tanto a nivel de aspecto gráfico como de interacción co usuario.

Como xa se explicou anteriormente, a especificación do formato de presentación do *ebook* é independente do proceso de edición do seu contido. Mais si existe certa relación entre o formato de presentación e o de publicación, xa que o modo no que un contido se presenta vai depender das capacidades que ofreza o formato no que se exporte finalmente o *ebook*. Por exemplo, un exercicio pode exportarse como un exercicio con corrección automática se o formato de exportación ofrece certas capacidades interactivas (por exemplo, o formato HTML5), mais isto non é posible se o formato de exportación non dispón desa capacidade (por exemplo, o formato EPUB 2.0). Neste caso concreto, a potencia de E-ditor está en que en ambos os dous casos é capaz de incorporar o exercicio ao *ebook*, como un exercicio con corrección automática ao exportar a un formato interactivo, ou como un enunciado seguido dunha ligazón a outra páxina coa solución, no caso dun formato que non soporta tanta interacción.

Debido a isto, E-ditor agrupa as funcionalidades deste módulo coas do Módulo de Exportación que se explica a continuación, a través do concepto de *acabado*. Un acabado representa un formato de exportación e unha plantilla (*template*) de presentación, que define tanto o aspecto gráfico que se aplicará aos contidos como a programación das posibles interaccións que o *ebooks* vai ofrecer ao usuario. E-ditor permite crear diferentes acabados para un mesmo contido, elixindo un dos formatos

de exportación soportados pola ferramenta e unha das plantillas existentes para ese formato.

Ademais, o Módulo de Presentación tamén permite a edición das plantillas, o que permite modificar por completo o aspecto dunha publicación. Isto é posible en acabados que empregan a linguaxe CSS para a definición dos estilos de presentación, e nos que se ofrece acceso a un editor textual sinxelo que, non obstante, require de certos coñecementos desta linguaxe, polo que se considera unha funcionalidade só para usuarios avanzados.

Módulo de Exportación

Este módulo é o encargado de xerar os recursos e empaquetalos nun arquivo no formato de distribución definitivo do *ebook*, a partir da información almacenada na base de datos e do formato de presentación definido a través do Módulo de Presentación. Os formatos aos que permite exportar na actualidade E-ditor son:

- EPUB. O formato estándar para a publicación de *ebooks* con maior soporte en diferentes dispositivos e máis empregado na actualidade. É o formato preferido para a publicación de *ebooks* literarios. Actualmente E-ditor exporta os *ebooks* en formato EPUB 2[12], que non soporta elementos interactivos, polo que un dos obxectivos no que xa estamos a traballar é na incorporación da última especificación do estándar (EPUB 3.0[13])
- SCORM[14]. O formato máis empregado para a distribución de contidos de *e-learning*, e que soportan a maioría de sistemas de xestión da aprendizaxe (LMS, *Learning Management Systems*).
- Web (HTML5). Os contidos empaquétanse nun arquivo ZIP que contén distintos recursos en formato web, entre os que se inclúen arquivos HTML, CSS, JavaScript, imaxes, vídeos etc., e que polo tanto pode ser visualizado desde calquera dispositivo que dispoña dun navegador web (PC, *smartphone*, *tablet* etc.). Este tipo de contido permite explotar a interactividade dos contidos web, polo que resulta moi axeitado para a publicación de *ebooks* enriquecidos sobre todo destinados a educación.

  Aínda que non existe un estándar para o empaquetado dun contido deste tipo, a fin de permitir a visualización dos *ebooks* neste formato do modo máis uniforme posible, inclúese no raíz do paquete un arquivo "index.html" que funcionará como lanzador, dando acceso ao resto de contidos do *ebook* a través de navegación con hiperenlaces.

[12] `http://idpf.org/epub/201`
[13] `http://idpf.org/epub/30`
[14] `http://www.adlnet.org/scorm.html`

Para rematar, indicar que os arquivos en formato HTML seguen a especificación da última revisión do estándar (HTML5), o que permite incorporar elementos como "`<video>`" e "`<audio>`" para a reprodución nativa destes formatos (sen depender de tecnoloxías propietarias de terceiros), ou o elemento "`<canvas>`" que permite a incorporación de gráficos dinámicos mediante programación con JavaScript.

- Android. Xérase un *ebooks* en formato APK (*Application Package File*), que é unha variante do formato JAR de Java para a distribución de aplicacións Android. Para a creación do APK, E-ditor xera o contido do libro no formato Web e embébeo nunha aplicación base que incorpora un compoñente navegador web (*WebView*[15]) para a súa visualización. A aplicación base, que en realidade fai as veces de plantilla de presentación, pode incorporar outras funcionalidades a maiores da simple visualización do contido, e que polo tanto estarán dispoñibles para todos os *ebooks* que se exporten con ela, como a posibilidade de incorporar anotacións ou marcadores.

Ademais da funcionalidade de exportar, que orixina o empaquetado e descarga do *ebook* xerado, este módulo permite tamén realizar unha previsualización da publicación desde unha interface incorporada ao propio espazo de traballo, o que resulta de moita axuda durante o proceso de creación.

8.1.2. Fluxo de traballo (*workflow*)

Atendendo ás distintas fases contempladas no proceso de edición dunha publicación dixital, E-ditor establece un fluxo de traballo que guía ao usuario durante dito proceso, que vai desde o rexistro da publicación introducindo a súa información descritiva até a xeración do arquivo de distribución. As fases deste fluxo de traballo na ferramenta, e que se explican en detalle máis adiante nesta sección, denomínanse "1.Información", "2.Estrutura", "3.Contidos" e "4.Maquetación".

As interfaces de usuario da ferramenta foron especialmente deseñadas para dar soporte a este fluxo de traballo, establecendo a secuencia lóxica que debe seguir o proceso de creación da publicación, mais permitindo ao mesmo tempo, acceder ás funcionalidades das fases posteriores. Non obstante, a ferramenta comproba en todo momento as posibles dependencias de cada operación con respecto a outras operacións de fases previas (como por exemplo que se intenta exportar unha publicación na fase de "4.Maquetación" da que non hai contidos xerados, ou que se intenta crear o índice de contidos na fase "3.Contidos" sen ter definida a estrutura da publicación na fase "2.Estrutura") e informa ao usuario cando non se pode levar a cabo a operación solicitada e por que motivo. O máis habitual será que tanto esta fase como a de

[15] `http://developer.android.com/reference/android/webkit/WebView.html`

"1.Información" se completen ao inicio do proceso de creación, investindo logo a maior parte do tempo nas fases "3.Contidos" e "4.Maquetación", sendo infrecuente ter que volver a elas.

Antes de entrar no proceso en si de creación dunha publicación dixital en E-ditor indicar que, unha vez autenticado no sistema, o primeiro ao que accede un usuario é a vista do seu catálogo de publicacións (denominados "materiais"). Esta vista permite ver as publicacións existentes, duplicar unha publicación, crear novas, eliminalas e organizalas xerarquicamente en carpetas, de forma semellante a como se faría nun sistema de arquivos gráfico. Tamén desde o catálogo de publicacións vai poder accederse ao espazo de edición da publicación, que se organiza nas seguintes fases que definen o fluxo de traballo:

1. Información. A primeira fase do proceso de creación dun contido dixital en E-ditor consisten na introdución da súa información descritiva e de catalogación, como o título, a autoría, o idioma, a data de creación, a imaxe da portada, a materia etc. De cara a ofrecer unha maior flexibilidade, só se considera obrigatorio o título da publicación, mais recoméndase cubrir todos aqueles campos que poidan ofrecer información de interese para o usuario consumidor.

   Esta fase está soportada polo submódulo Catálogo do Módulo de Edición.

2. Estrutura. A segunda fase é a da definición da estrutura do material que, como xa comentamos, representa a un nivel alto de abstracción o modo no que se organizan os contidos do material. Para facilitar este paso, ofrécense algunhas estruturas por defecto (para un libro de texto, unha unidade didáctica, unha creación literaria etc.) que resultan axeitadas para as publicacións máis habituais. Para o caso de precisar unha estrutura diferente, pode definirse desde cero ou modificando unha existente a través dun sinxelo editor gráfico de estruturas, que permite crear a xerarquía de elementos estruturais (temas, capítulos, subcapítulos, apartados etc.) e tipos de contido (texto, exercicios, ligazóns etc.) mediante funcionalidades de arrastrar e soltar. Por último, indicar que tamén se permite reutilizar a estrutura doutra publicación existente, o que resulta moi útil no caso de publicacións cunha organización dos contidos homoxénea.

   As funcionalidades desta fase son ofrecidas polo submódulo Definición da Estrutura do Módulo de Edición.

3. Contidos. A seguinte fase é a de introdución dos contidos propiamente dito. Trátase da fase de traballo principal, e que consumirá a maior parte dos esforzos, xa que é aquela na que se van introducir os contidos (textos, imaxes, vídeos, sons, actividades interactivas etc.) que forman o material.

   A creación dos contidos parte do índice, inicialmente baleiro, no que se irán engadindo os temas, capítulos, apartados etc. que forman parte da publicación,

e seguindo a organización definida na estrutura. Dentro de cada un destes será posible incluír os elementos de contido do tipo permitido tamén na estrutura da publicación. Para cada tipo de contido, E-ditor dispón dunha ferramenta de edición específica, como o editor WYSIWYM para os contidos conceptuais (texto), ou un editor de exercicios que ofrece as interfaces precisas para a creación de cada un dos tipos de exercicios soportados.

Esta fase está soportada polo submódulo Edición de Contidos do Módulo de Edición.

4. Maquetación. A última fase do proceso creativo, e que permite definir os acabados (plantilla de estilo e formato de publicación) cos que se exportarán os *ebooks*. Para cada publicación é posible crear distintos acabados (mesmo varios acabados para un mesmo formato de exportación) que permitirán obter diferentes versións do *ebook*.

   As funcionalidades dispoñibles para esta fase son responsabilidade conxunta de dous módulos, o Módulo de Presentación e o Módulo de Exportación.

Parte III

Conclusións e retos futuros

Capítulo 9

Conclusións

Neste capítulo recollemos as conclusións que se derivan da investigación e desenvolvementos presentados ao longo da tese.

Na primeira parte da tese amosamos como os nosos autoíndices construídos sobre textos en linguaxe natural que consideran a ditos textos como secuencias de palabras en vez de caracteres representan unha alternativa moi interesante aos índices invertidos, sobre todo en contextos nos que o espazo ocupado polos textos e índices é crítico, pero nos que á vez se precisa manter unha boa eficiencia nas operacións de busca e descompresión. Así, comprobamos como empregando menos do 40 % do espazo ocupado polo texto orixinal, os índices invertidos non poden competir en eficiencia cos autoíndices de palabras. Empregando unha cantidade de espazo maior, os nosos autoíndices seguen sendo en xeral máis rápidos que os índices invertidos para localizar as ocorrencias de palabras individuais ou frases (no caso das frases, a mellora é notable), e para extraer fragmentos de textos ao redor de ocorrencias de frases.

Tamén amosamos como é posible incorporar unha capa de presentación sobre o índice (`FWCSA`) para permitir ampliar as procuras a outras operacións típicas en Recuperación de Información como a busca por raíces de palabras, omitindo *stopwords*, sen facer distinción entre maiúsculas e minúsculas etc., obtendo unha eficiencia espacial e temporal comparable á dos autoíndices orixinais (`WCSA`).

Por último comprobamos que, mesmo como técnica de compresión, os nosos autoíndices de palabras permiten obter porcentaxes de compresión comparables ás de moitos dos compresores de texto para linguaxe natural, e que non dispoñen de funcionalidades de indexación. Así, vimos que tanto `WCSA` como `FWCSA` permiten obtén porcentaxes de compresión da orde do 35–40 % do tamaño orixinal do texto,

e que incluso poden alcanzar valores próximos ao 30 %, aínda que penalizando a eficiencia nas buscas.

Tamén na primeira parte da tese amosamos como é posible empregar a información que ofrece o LCP (*Longest Common Prefix*) do array de sufixos orientado a palabras dun texto, para desenvolver unha técnica de compresión "variable-a-variable" que considera frases como símbolos no vocabulario do compresor, e sobre as que aplica unha codificación estatística, concretamente `ETDC`. Tamén vimos como o emprego de diferentes heurísticas para a selección das frases do vocabulario permite desenvolver distintas variantes (`V2VDC` e $\texttt{V2VDC}_H$) que ofrecen á súa vez distinto rendemento en canto a porcentaxe de compresión e, sobre todo, a tempo de compresión/descompresión.

Demostramos como a nosa técnica se sitúa entre as mellores do estado da cuestión en compresores de texto. Obtén boas porcentaxes de compresión, xa que supera lixeiramente a `bzip2` e obtén resultados próximos aos de `ppmdi`, `p7zip` e `re-pair`. A nosa variante máis rápida `V2VDC` é máis lenta que `bzip2` na compresión e obtén resultados similares a `ppmdi`, mentres que supera de forma significativa os tempos obtidos por `re-pair` e `p7zip`. No referente á descompresión, ningunha técnica pode competir cos resultados das nosas variantes de `V2VDC`, que son habitualmente o dobre de rápidas que `p7zip` e `re-pair`, tres veces máis rápidas que `bzip2`, e entre 20 e 30 veces máis rápidas que `ppmdi`.

Tamén comparamos as variantes de `V2VDC` con outras técnicas de compresión máis rápidas pero que ofrecen peores porcentaxes de compresión, comprobando por exemplo que as nosas propostas superan a `gzip` en aproximadamente 10 puntos porcentuais, e ao compresor de texto semiestático orientado a palabra `ETDC` por 7–10 puntos. Como era de esperar, o novo compresor é máis lento ao comprimir que `ETDC` e entre 3 e 4 veces máis lento que `gzip`. Con todo, as cousas cambian na descompresión, onde a nosa proposta obtén resultados parellos aos da técnica `ETDC`.

Por último, demostramos como é posible realizar buscas eficientes directamente sobre o texto comprimido con `V2VDC`, superando mesmo ao `ETDC` (agás para patróns moi frecuentes), o que é moi destacable se temos en conta que `ETDC` era até o momento a técnica de compresión que permitía buscar máis rápido no texto comprimido.

Xa na segunda parte da tese, amosamos distintas solucións tecnolóxicas que desenvolvemos para resolver diferentes aspectos ao redor do consumo de contidos dixitais, e demostramos como estas solucións resultaron efectivas na práctica mediante a súa incorporación en plataformas de distribución comerciais que están actualmente en produción. Tamén vimos a utilidade de empregar estruturas de datos compactas e algoritmos avanzados, resultado da investigación en compresión e indexación de texto en linguaxe natural, nestas solucións, que obteñen un beneficio directo da

mellora na eficiencia espacial e temporal que estas técnicas ofrecen nas operacións de almacenamento, acceso, procesado e transmisión do texto.

Por último, demostramos como é posible abordar o proceso de creación de *ebooks* desde unha perspectiva completamente diferente e innovadora, baseada na separación dos diferentes aspectos que forman parte dunha publicación dixital (a estrutura dos contidos, os propios contidos, o estilo de presentación e o formato de publicación), e amosamos as vantaxes que esta separación permite obter. Sobre esta idea, presentamos E-ditor, unha aplicación SaaS para a creación de contidos, dirixida fundamentalmente a editoras, que establece un *workflow* que guía o proceso de creación, e que permite introducir os contidos dunha publicación unha única vez e exportala a diferentes formatos de *ebook*.

Capítulo 10

Retos futuros

Nesta sección imos introducir algunhas das liñas de traballo abertas tanto na investigación en estruturas de datos e algoritmos para compresión e indexación de texto, correspondente á primeira parte desta tese, como ao desenvolvemento de solucións tecnolóxicas orientadas a resolver diferentes aspectos do negocio dos contidos dixitais, especialmente dirixidas ao sector editorial, e que derivan dos desenvolvementos presentados na segunda parte da tese. Ademais, na área dos contidos dixitais, a velocidade dos avances tecnolóxicos, tanto a nivel de dispositivos como de formatos de publicación, fai necesario un seguimento tecnolóxico e de mercado continuo que permita ofrecer unha resposta a tempo ás novas demandas dos usuarios.

Autoíndices orientados a palabras

Como liña futura contemplamos estudar a viabilidade dos autoíndices para resolver outras funcionalidades que ofrecen os índices invertidos, como as explicadas en [BYRN99]. Por exemplo os índices invertidos utilízanse para implementar o modelo tf-idf gardando o número de ocorrencias de cada palabra en cada vocabulario, ordenadas por frecuencia. Así só é necesaria a recuperación dun prefixo curto das listas para resolver as consultas. Sadakane [Sad07] ten presentado algúns avances para poder utilizar o `CSA` de forma semellante, pero é necesario un estudo máis minucioso.

Ademais, esta contribución baséase nun autoíndice en particular, o `CSA` de Sadakane, polo que a mesma aproximación podería seguirse para crear outros autoíndices que tamén teñen dependencia do tamaño do vocabulario. Na versión de revista do noso artigo en [FBN+12] xa vimos como adaptar un *Succint Suffix*

Array para crear unha estrutura similar ao `WCSA`, mais os resultados non foron en xeral tan bos como os acadados polo `WCSA`. Pensamos que tamén podería ser interesante adaptar o LZ-index [Nav04, ANS06] para ofrecer unha rápida localización de ocorrencias.

Compresión orientada a frases

Como traballo futuro contemplamos o estudo de novas heurísticas que poidan derivar nunha mellor selección de frases. Tamén sería interesante optimizar os cálculos das estatísticas necesarias para esas heurísticas. Alén diso, a construción do *LCP* en disco é un problema de investigación aberto [KSB06] que permitiría comprimir corpus máis grandes de forma eficiente coa nosa proposta.

Distribución de contidos dixitais

Un dos traballos futuros máis inmediatos que temos previsto é o desenvolvemento das *apps* móbiles Xcloud Reader para Android e iOS. Como vimos, o visor Xcloud Reader que desenvolvemos para a lectura de *ebooks* na modalidade de *streaming* é totalmente accesible desde dispositivos móbiles, ao estar desenvolvido empregando tecnoloxías web estándar (HTML5). Non obstante, as aplicacións móbiles nativas resultan en xeral máis atractivas para os usuarios de *smartphones* e *tablets* que as web, ao ofrecer un uso máis fluído e unha interface tamén máis atractiva.

Creación de contidos

Na actualidade, E-ditor ofrece unha solución completamente funcional e suficiente para que editoras, centros educativos, docentes e demais entidades ou persoas para a creación dos seus propios contidos dixitais. Non obstante, para que a utilidade da ferramenta se manteña vixente no tempo ou facela máis atractiva a unha cantidade maior de usuarios, é imprescindible ir incorporando nela novas funcionalidades como, por exemplo, a posibilidade de crear novos tipos de contidos, ou o soporte de novos formatos de publicación e novas versións dos existentes.

Outro dos desenvolvementos que contemplamos abordar nun futuro próximo é a integración da ferramenta de creación de contidos E-ditor coa plataforma de distibución Xcloud Bookstore, para que as editoras poidan publicar de forma automática os *ebooks* creados con E-ditor na súa plataforma de venda.

Apéndice A

Publicacións e outros resultados de investigación

Este capítulo enumera as miñas publicacións e outros resultados de investigación directamente relacionadas con esta tese.

Publicacións

Revistas

- Fariña, A.; Brisaboa, N. R.; Navarro, G.; Claude, F.; Places, A. S.; Rodríguez López, E.: Word-based Self-Indexes for Natural Language Text. *ACM Transactions on Information Systems (TOIS)*, 30(1), pp. 1–34, 2012.
  - Revista JCR indexada no Q2 de "Computer Science, Information Systems".
  - Artigo citado 22 veces (en setembro de 2015 segundo Google Scholar).
- P-Sanjulián, C. F.; Pérez R., M.A.; Rodríguez López, E.; Places, A. S.: Recursos clasificación da produción editorial na Galiza durante o franquismo: deseño e alimentación da base de datos. *Janus* (Anexo I), pp. 161–174, 2014.
- Baranda, C.; Rodríguez López, E.: Red ARACNE: retos y objetivos de un proyecto de coordinación en letras hispánicas digitales. *Janus* (Anexo I), pp. 101–109, 2014.

Capítulos de libro

- Rodríguez López, E.; Places, A. S.: e-ditor: Herramienta de autor en la nube para la creación de libros electrónicos. *Visibilidad y divulgación de la investigación desde las Humanidades Digitales. Experiencias y proyectos*, Baraibar, Álvaro (ed.), pp. 205–221, 2014.
- Arrigoni E.; Rodríguez López, E.: La red de investigación de Humanidades Digitales y Letras Hispánicas: avance de Red-ARACNE. *Visibilidad y divulgación de la investigación desde las Humanidades Digitales. Experiencias y proyectos*, Baraibar, Álvaro (ed.), pp. 243–251, 2014.

Conferencias internacionais

- Brisaboa, N. R.; Fariña, A.; López, J. R.; Navarro, G.; Rodríguez López, E.: A New Searchable Variable-to-Variable Compressor. *Proceedings of the 20th Data Compression Conference (DCC 2010)*, IEEE Computer Society, pp. 199–208, 2010.
  - Conferencia indexada como CORE A*.
  - Artigo citado 11 veces (en setembro de 2015 segundo Google Scholar).
- Brisaboa, N. R.; Fariña, A.; Navarro, G.; Places, A. S.; Rodríguez López, E.: Self-indexing Natural Language. *Proceedings of the 15th International Symposium on String Processing and Information Retrieval (SPIRE 2008)*, LNCS 5280, Springer Verlag, pp. 121–132, 2008.
  - Conferencia indexada como CORE B.
  - Artigo citado 18 veces (en setembro de 2015 segundo Google Scholar).
- Rodríguez López, E.; Places, A. S.; Cotelo, J. A.; Pedreira, O.; Brisaboa, N. R.: Towards Commercial eBook Production in Small Publishing Houses. *Proceedings of the 5th International Conference on Computer Supported Education (CSEDU 2013)*, SCITEPRESS, pp. 116–121, 2013.

Conferencias nacionais

- Places, A. S.; Rodríguez López, E.: BIDISO. Biblioteca Digital Siglo de Oro. *Actas del Seminario Internacional sobre BIDESLITE*, Madrid, pp. 74–87, 2011.
- Brisaboa, N. R.; Fariña, A.; Ladra, S.; Places, A. S.; Rodríguez López, E.: Indexación y autoindexación comprimida de documentos como base de su procesado. *Actas del I Congreso Español de Recuperación de Información (CERI 2010)*, Madrid, pp. 137–148, 2010.

- Lamas, J.I.; Luaces, M. R.; Places, A. S.; Rodríguez López, E.; Seco, D.: Los GIS y servicios IDE en la difusión de la cultura gallega. La Web Cultura Galega. *Actas de las V Jornadas de la Infraestructura de Datos Espaciales de España (JIDEE 2008)*, Tenerife, 2008.

- Rodríguez López, E.; Fariña, A.; Places, A. S.; Paramá, J. R.; Pedreira, O.: WCSA: Un autoíndice orientado a palabras para textos en lenguaje natural. *Actas de las XII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2007)*, Zaragoza, pp. 144–153, 2007.

Rexistro de software

- *E-ditor*. Eduardo Rodriguez López, Andrés Basoa, Ángeles Saavedra Places, Nieves R. Brisaboa, Óscar Pedreira e Miguel R. Luaces. Data de concesión: 02/01/2014.
  Empresas que o explotan: Enxenio S.L.

Bibliografía

[AL00] A. Apostolico e S. Lonardi. Off-line compression by greedy textual substitution. *Proceedings of the IEEE (PIEEE)*, 88(11):1733–1744, 2000.

[AM05] V. Anh e A. Moffat. Inverted index compression using word-aligned binary codes. *Information Retrieval (IR)*, 8:151–166, 2005.

[AM10] V. Anh e A. Moffat. Index compression using 64-bit words. *Software Practice and Experience (SPE)*, 40(2):131–147, 2010.

[ANS06] D. Arroyuelo, G. Navarro, e K. Sadakane. Reducing the space requirement of LZ-index. In *Proc. of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 319–330, 2006.

[BCW90] T. C. Bell, J. G. Cleary, e I. H. Witten. *Text Compression.* Prentice Hall, 1990.

[BFL+10] N. Brisaboa, A. Fariña, J. R. López, G. Navarro, e E. Rodríguez López. A new searchable variable-to-variable compressor. In *Proc. of the 20th Data Compression Conference (DCC)*, pages 199–208, 2010.

[BFLN08] N. Brisaboa, A. Fariña, S. Ladra, e G. Navarro. Reorganizing compressed text. In *Proc. of the 31st Annual International ACM SIGIR Conference on Research and Development on Information Retrieval (SIGIR)*, pages 139–146, 2008.

[BFLN12] N. Brisaboa, A. Fariña, S. Ladra, e G. Navarro. Implicit indexing of natural language text by reorganizing bytecodes. *Information Retrieval (IR)*, 16(6):527–557, 2012.

[BFN+08] N. Brisaboa, A. Fariña, G. Navarro, A. S. Places, e E. Rodríguez López. Self-indexing natural language. In *Proc. of the 15th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 121–132, 2008.

[BFNP04] N. Brisaboa, A. Fariña, G. Navarro, e J. Paramá. Simple, fast, and efficient natural language adaptive compression. In *Proc. of the 11th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 230–241, 2004.

[BFNP05] N. Brisaboa, A. Fariña, G. Navarro, e J. Paramá. Efficiently decodable and searchable natural language adaptive compression. In *Proc. of the 28th Annual International ACM SIGIR Conference on Research and Development on Information Retrieval (SIGIR)*, pages 234–241, 2005.

[BFNP07] N. Brisaboa, A. Fariña, G. Navarro, e J. Paramá. Lightweight natural language text compression. *Information Retrieval (IR)*, 10:1–33, 2007.

[BFNP08] N. Brisaboa, A. Fariña, G. Navarro, e J. Paramá. New adaptive compressors for natural language text. *Software: Practice and Experience (SPE)*, 38(13):1429–1450, 2008.

[BFNP10] N. Brisaboa, A. Fariña, G. Navarro, e J. Paramá. Dynamic lightweight text compression. *ACM Transactions on Information Systems (TOIS)*, pages 1–32, 2010.

[BINP03] N. Brisaboa, E. Iglesias, G. Navarro, e J. Paramá. An efficient compression code for text databases. In *Proc. of the 25th European Conference on Information Retrieval Research (ECIR)*, pages 468–481, 2003.

[BK02] J. Barbay e C. Kenyon. Adaptive intersection and t-threshold problems. In *Proc. of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–399, 2002.

[BLLS09] J. Barbay, A. López-Ortiz, T. Lu, e A. Salinger. An experimental investigation of set intersection algorithms for text searching. *ACM Journal of Experimental Algorithmics (JEA)*, 14:artigo 7, 2009.

[BLOL06] J. Barbay, A. López-Ortiz, e T. Lu. Faster adaptive set intersections for text searching. In *Proc. of the 5th International Workshop on Experimental Algorithms (SEA)*, pages 146–157, 2006.

[BM77] R. Boyer e J. Moore. A fast string searching algorithm. *Communications of the ACM (CACM)*, 20(10):762–772, 1977.

[BSTW86] J. Bentley, D. Sleator, R. Tarjan, e V. Wei. A locally adaptive data compression scheme. *Communications of the ACM (CACM)*, 29(4):320–330, 1986.

[BY04] R. Baeza-Yates. A fast set intersection algorithm for sorted sequences. In *Proc. of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 400–408, 2004.

[BYRN99] R. Baeza-Yates e B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[CLL+05] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, e A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory (TIT)*, 51(7):2554–2576, 2005.

[CM07] J. Culpepper e A. Moffat. Compact set representation for information retrieval. In *Proc. of the 14th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 137–148, 2007.

[CM10] J. Culpepper e A. Moffat. Efficient set intersection for inverted indexing. *ACM Transactions on Information Systems (TOIS)*, 29(1):artigo 1, 2010.

[CN08] F. Claude e G. Navarro. Practical rank/select queries over arbitrary sequences. In *Proc. of the 15th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 176–187, 2008.

[CRM91] S. Card, G. Robertson, e J. Mackinlay. The information visualizer, an information workspace. In *Proc. of the 9th Conference on Human Factors in Computing Systems (SIGCHI)*, pages 181–186, 1991.

[CW84] J. G. Cleary e I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications (TCOM)*, 32(4):396–402, 1984.

[Deu96] P. Deutsch. Deflate compressed data format specification version 1.3, 1996.

[DM00] E. Demaine e I. Munro. Adaptive set intersections, unions, and differences. In *Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.

[FBN+12] A. Fariña, N. Brisaboa, G. Navarro, F. Claude, A. S. Places, e E. Rodríguez López. Word-based self-indexes for natural language text. *ACM Transactions on Information Systems (TOIS)*, pages 1–34, 2012.

[FM00] P. Ferragina e G. Manzini. Opportunistic data structures with applications. In *Proc. of the 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 390–398, 2000.

[FM05] P. Ferragina e G. Manzini. Indexing compressed texts. *Journal of the ACM (JACM)*, 52(4):552–581, 2005.

[FMMN07] P. Ferragina, G. Manzini, V. Mäkinen, e G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms (TALG)*, 3(2):artigo 20, 2007.

[FNP08] A. Fariña, G. Navarro, e J. Paramá. Word-based statistical compressors as natural language compression boosters. In *Proc. of the 18th Conference on Data Compression (DCC)*, pages 162–171, 2008.

[Gal78] R.G Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory (TIT)*, 24(6):668–674, 1978.

[GGV03] R. Grossi, A. Gupta, e J. Vitter. High-order entropy-compressed text indexes. In *Proc. of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 841–850, 2003.

[GV00] R. Grossi e J. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proc. of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 397–406, 2000.

[Hea78] H. Heaps. *Information Retrieval - Computational and Theoretical Aspects*. Academic Press, NY, 1978.

[Hem05] S. Heman. *Super-scalar database compression between RAM and CPU-cache*. PhD thesis, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 2005.

[Hor80] R. N. Horspool. Practical fast searching in strings. *Software: Practice and Experience (SPE)*, 10(6):501–506, 1980.

[Huf52] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE (PIRE)*, 40(9):1090–1101, 1952.

[KLA+01] T. Kasai, G. Lee, H. Arimura, S. Arikawa, e K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proc. of the 12th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 181–192, 2001.

[KSB06] J. Kärkkäinen, P. Sanders, e S. Burkhardt. Linear work suffix array construction. *Journal of the ACM (JACM)*, 53(6):918–936, 2006.

[LBK15] D. Lemire, L. Boytsov, e N. Kurz. SIMD compression and the intersection of sorted integers. Technical report, arXiv:1401.6399, 2015.

[LM00] J. Larsson e A. Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE (PIEEE)*, 88(11):1722–1732, 2000.

[Mäk00] V Mäkinen. Compact suffix array. In *Proc. of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 305–319, 2000.

[MC07] A. Moffat e S. Culppeper. Hybrid bitvector index compression. In *Proc. of the 12th Annual Australasian Document Computing Symposium (ADCS)*, pages 25–31, 2007.

[MF04] G. Manzini e P. Ferragina. Engineering a lightweight suffix array construction algorithm. *Algorithmica (ALGO)*, 40(1):33–50, 2004.

[Mil68] R. Miller. Response time in man-computer conversational transactions. In *Proc. of the December 9-11, 1968, Fall Joint Computer Conference, Part I (AFIPS)*, pages 267–277, 1968.

[MK95] A. Moffat e J. Katajainen. In-place calculation of minimum-redundancy codes. In *Proc. of the 4th Workshop on Algorithms and Data Structures (WADS)*, pages 393–402, 1995.

[MM90] U. Manber e G. Myers. Suffix arrays: A new method for on-line string searches. In *Proc. of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 319–327, 1990.

[MM93] U. Manber e G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing (SICOMP)*, 22(5):935–948, 1993.

[MN05] V. Mäkinen e G. Navarro. Succinct suffix arrays based on run-length encoding. *Nordic Journal of Computing (NJC)*, 12(1):40–66, 2005.

[MN08] V. Mäkinen e G. Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Transactions on Algorithms (TALG)*, 4(3):artigo 32, 2008.

[MNZB98] E. Moura, G. Navarro, N. Ziviani, e R. Baeza-Yates. Fast searching on compressed text allowing errors. In *Proc. of the 21st Annual International ACM SIGIR Conference on Research and Development on Information Retrieval (SIGIR)*, pages 298–306, 1998.

[MNZBY00] E. Moura, G. Navarro, N. Ziviani, e R. Baeza-Yates. Fast and flexible word searching on compressed text. *ACM Transactions on Information Systems (TOIS)*, 18(2):113–139, 2000.

[Mof89] A. Moffat. Word-based text compression. *Software: Practice and Experience (SPE)*, 19(2):185–198, 1989.

[Nav04] G. Navarro. Indexing text using the Ziv-Lempel trie. *Journal of Discrete Algorithms (JDA)*, 2(1):87–114, 2004.

[Nie93] J. Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[NM07] G. Navarro e V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys (CSUR)*, 39(1):artigo 2, 2007.

[NMN+00] G. Navarro, E. Moura, M. Neubert, N. Ziviani, e R. Baeza-Yates. Adding compression to block addressing inverted indexes. *Information Retrieval (IR)*, 3(1):49–77, 2000.

[NMWM94] C. Nevill-Manning, I. Witten, e D. Maulsby. Compression by induction of hierarchical grammars. In *Proc. of the 4th Conference on Data Compression (DCC)*, pages 244–253, 1994.

[NR02] G. Navarro e M. Raffinot. *Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002.

[OV14] G. Ottaviano e R. Venturini. Partitioned Elias-Fano indexes. In *Proc. of the 37th Annual International ACM SIGIR Conference on Research and Development on Information Retrieval (SIGIR)*, pages 273–282, 2014.

[RLPC+13] E. Rodríguez López, A. S. Places, J. A. Cotelo, O. Pedreira, e N. Brisaboa. Towards commercial ebook production in small publishing houses. In *Proc. of the 5th International Conference on Computer Supported Education (CSEDU)*, pages 116–121, 2013.

[Ryt03] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science (TCS)*, 302(1-3):211–222, 2003.

[Sad03] K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *Journal of Algorithms (JAL)*, 48(2):294–313, 2003.

[Sad07] K. Sadakane. Succinct data structures for flexible text retrieval systems. *Journal of Discrete Algorithms (JDA)*, 5(1):12–22, 2007.

[SC07] T. Strohman e B. Croft. Efficient document retrieval in main memory. In *Proc. of the 30th Annual International ACM SIGIR Conference on Research and Development on Information Retrieval (SIGIR)*, pages 175–182, 2007.

[SGL10] B. Schlegel, R. Gemulla, e W. Lehner. Fast integer compression using SIMD instructions. In *Proc. 6th Int. Workshop on Data Management on New Hardware (DaMon)*, pages 34–40, 2010.

[SGR+11] A. Stepanov, A. Gangolli, D. Rose, R. Ernst, e P. Oberoi. SIMD-based decoding of posting lists. In *Proc. of the 20th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 317–326, 2011.

[Sha01] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R)*, 5(1):3–55, January 2001.

[ST07] P. Sanders e F. Transier. Intersection in integer inverted indices. In *Proc. 9th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 71–83, 2007.

[ST08] P. Sanders e F. Transier. Compressed inverted indexes for in-memory search engines. In *Proc. of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 3–12, 2008.

[Tro14] A. Trotman. Compression, simd, and postings lists. In *Proc. of the 2014 Australasian Document Computing Symposium (ADCS)*, pages 50–57, 2014.

[TS10] F. Transier e P. Sanders. Engineering basic algorithms of an in-memory text search engine. *ACM Transactions on Information Systems (TOIS)*, 29:artigo 2, 2010.

[Vit87] J.S. Vitter. Design and analysis of dynamic Huffman codes. *Journal of the ACM (JACM)*, 34(4):825–845, 1987.

[WMB99] I. Witten, A. Moffat, e T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, second edition, 1999.

[WZ99] H. E. Williams e J. Zobel. Compressing integers for fast file access. *The Computer Journal (COMPJ)*, 42(3):193–201, 1999.

[ZHNB06] M. Zukowski, S. Heman, N. Nes, e P. Boncz. Super-scalar RAM-CPU cache compression. In *Proc. of the 22nd IEEE International Conference on Data Engineering (ICDE)*, page 59, 2006.

[ZL77] J. Ziv e A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory (TIT)*, 23(3):337–343, 1977.

[ZL78] J. Ziv e A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory (TIT)*, 24(5):530–536, 1978.

[ZM06] J. Zobel e A. Moffat. Inverted files for text search engines. *ACM Computing Surveys (CSUR)*, 38(2)(2):artigo 6, 2006.