UNIVERSIDAD DE OVIEDO

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

"MERA: Musical Entities Reconciliation Architecture"

DIRECTOR: Jose Emilio Labra Gayo

AUTOR: Daniel Fernández Álvarez

Vº Bº del Director del Proyecto

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Agradecimientos

No hubiese sido posible afrontar este proyecto sin la intervención de muchas personas.

Agradezco a mis compañeros en el Laboratorio de Tecnologías Orientadas a Objetos de la Facultad de Ciencias el ambiente de camaradería y ayuda desinteresada que siempre ha habido, en especial a Cristian por su continuo apoyo y a Ricardo por todo el tiempo que hemos echado encerrados trabajando a horas intempestivas.

Agradezco a mi familia, principalmente a mis padres y hermana, su comprensión y ayuda en todo lo imaginable, durante este trabajo y durante toda mi vida.

Agradezco Cristina, mi novia, su apoyo, su paciencia, su cariño incondicional, su esfuerzo para que todo vaya bien y toda la fuerza que me da. Además, le agradezco su aportación con el inglés en este artículo.

Agradezco a mis amigos de siempre, que han sabido disculpar mis ausencias prolongadas haciéndome sentir que ni la distancia ni los diferentes caminos que vamos tomando pueden con lo que nos une.

Agradezco a mis compañeros en el Master de Ingeniería Web el camino andado juntos, el buen ambiente y el esfuerzo brutal que hemos conseguido realizar en muchas ocasiones, siempre más llevadero con el apoyo mutuo.

Agradezco al Profesor Daniel Gayo su atención y ayuda en diferentes etapas de este trabajo y a mi director de proyecto, Jose Emilio Labra, su confianza ciega.

A todos ellos, muchísimas gracias.

Capítulo 2

Resumen

El problema de la reconciliación de entidades (record linkage) ha sido profundamente estudiado a pesar del hecho de ser aún hoy en día una rama de investigación abierta y activa. La proliferación de bases de datos con grandes volúmenes de datos de diversa naturaleza por la World Wide Web conduce a un interés generalizado de encontrar técnicas precisas y eficientes de detección de entradas equivalentes en aquellos escenarios en los que no se dispone de identificadores únicos confiables. Esto es deseable tanto para la detección de duplicados dentro de una propia base de datos como para el reconocimiento de entradas en diferentes bases de datos que hacen referencia a la misma realidad.

Existe un conjunto de problemas habituales cuando nos enfrentamos a un problema de reconciliación de entidades, como puede ser la presencia de erratas. No obstante, los diferentes tipos de fuentes pueden presentar distintos tipos de problemas asociados a la hora de llevar a cabo un proceso de reconciliación. Hemos estudiado el caso concreto de las fuentes de datos asociadas al mundo musical, encontrando que no es posible elaborar una lista común de problemas asociados a todas las bases de datos de este tipo. La calidad de los datos, el idioma, las convenciones de nombres y el tipo de contenido son algunos de los factores que determinan qué tipo de estrategias se deberían poner en marcha en cada caso para llevar a cabo satisfactoriamente un proceso de conciliación. Hemos también observado que ciertos conceptos musicales son habitualmente referidos usando formas válidas pero de distinta naturaleza. Un ejemplo ilustrativo sería la forma de denominar a un artista o grupo en una fuente de datos concreta, donde tanto nombres artísticos, civiles, alias o componentes de un grupo podrían ser usados.

Para aportar una mejora sobre los actuales sistemas de reconocimiento de entidades musicales proponemos MERA (Musical Entities Reconciliation Architecture), una arquitectura pensada para el linkado de dos bases de datos capaz de adaptarse a las técnicas y los algoritmos de reconciliación más convenientes en cada caso particular. MERA incluye ademas mecanismos para involucrar fuentes de datos de terceros con el objetivo de mejorar los resultados. Nuestra propuesta está basada en tecnologías de web semántica, almacenando y manejando información a través del uso de grafos rdf. MERA trabaja sobre un esquema general de relaciones entre los nodos del grafo para organizar la información, no necesariamente atado a ontologías particulares, proponiendo una nueva forma de exploración de grafo acorde a dicho esquema. Hemos implementado y evaluado un prototipo que recoge la mayor parte de los aspectos de diseño de MERA.

Capítulo 3

Abstract

The problem of entity recognition or record linkage has been largely studied despite the fact it is still an opened issue. The proliferation of large databases with potentially repeated records across the World Wide Web drives into a generalized interest to find precise and efficient methods to detect duplicate entries when no reliable unique identifiers are available. This is desirable both to detect duplicate records within a database and to recognize as pointers to the same real-world object two records in different databases.

There is a set of issues that may appear when facing a record linkage scenario that are potentially shared by databases or different nature, such as the presence of misspellings. However, the different types of sources may present different type of common issues. We have studied the cases of datasets containing information linked to the music world, finding that is hard to elaborate a list of problems common to every musical database. Data quality, naming conventions and content nature may drastically vary the list of issues to face in each case. We also find some musical concepts are usually named with different but valid forms that identify the same reality. An illustrative is the way to designate an artist or group, where artistic name, civil name(s), group members or alias could be provided.

We propose MERA (Musical Entities Recognition Architecture), an architecture thought to link two databases adapting the techniques and reconciliation algorithms to each particular case. MERA also includes mechanisms to manage third party sources to improve the results. Our approach is based in web semantic technologies, storing and organizing the information in rdf graphs, in which every entity could be linked to its related concepts or known alternative forms. MERA defines a graph schema to organize info, not ne-

cessary linked to particular ontologies, and propose a novel way no navigate that graph during the linking process. We have implemented and successfully evaluated a prototype that catch most of MERA's design aspects.

Capítulo 4

Palabras clave

Palabras clave:

- Linkado de entidades
- Reconciliación de metadatos musicales
- Arquitectura adaptable
- Grafo rdf

Keywords:

- Record linkage
- Music metadata reconciliation
- Adaptable architecture
- Rdf graph

Capítulo 5

Introducción

En este trabajo pretendemos aportar una mejora para la tarea específica del reconocimiento de entradas equivalentes en fuentes de datos relacionadas con el mundo de la música. Para ello presentamos MERA (Musical Entities Reconciliation Architecture), la arquitectura de un sistema diseñado para encontrar vínculos de equivalencia entre distintas elementos que sin embargo hacen referencia a la misma realidad. MERA está diseñada para poder adaptar el proceso de reconciliación a la naturaleza de los distintos escenarios concretos, tratando de alcanzar cuotas de precisión tan altas como sea posible.

Ejemplos de campos encontrados comúnmente en bases de datos relacionadas con el mundo de la música son títulos de canciones, nombres de artistas, álbumes, géneros, etc. La tarea de reconocer equivalencias entre este tipo de entradas está fuertemente conectada al problema de la *reconciliación de entidades*, puesto que consiste en la detección de elementos distintos que representan el mismo concepto del mundo real en situaciones en los que no se dispone de identificadores únicos confiables. No obstante, hemos diseñado MERA partiendo de la hipótesis de que el tipo de datos asociado al mundo de la música tiene asociadas una serie de peculiaridades específicas respecto a datos de otra naturaleza, que de ser tenidas en cuenta pueden mejorar el proceso de reconciliación respecto a tácticas pensadas para contextos más generales. Un ejemplo de las particularidades asociadas a este tipo de datos lo encontramos en la especificación de nombres de artistas. Son muchas las formas válidas o reconocibles usadas para referirse a cierto artista, y tratar de reconciliar conjuntos de datos que usen distintas formas para designar a un mismo artista puede resultar difícil. Algunas de esas formas serían:

- *Nombre artístico y nombre civil*: “Stefani Joanne Angelina Germanotta” y “Lady Gaga”.
- *Distintas convenciones de nombrado*:“The Beatles” o “Beatles, the”.
- *Alias oficiales o ampliamente extendidos*: “El Rey del Rock” en lugar de “Elvis Presley”.
- *Mezclas entre nombres artísticos y civiles*: “Shakira”, “Shakira Isabel Mebarack Ripoll”, “Shakira Mebarack”,...
- *Acrónimos*: “SOAD” en lugar de “System of a down”.
- *Errores comunes de escritura*: “Bruce Springsting” en lugar de “Bruce Springsteen”.
- *Metonimia*: uso del nombre de un artista cuando en realidad se debe hacer referencia el nombre de un grupo. Ejemplo: “Michael Jackson” en lugar de “The Jackson Five”.

Problemas como la posible presencia de erratas o errores ortográficos no son específicos del mundo de la música, sino que pueden darse en bases de datos de diversos campos. En cambio, la existencia de un nombre civil y un nombre artístico para especificar la misma entidad sí es un problema exclusivo de la denominación de artistas. Por contra, si tratamos de reconciliar otro tipo de entidad musical como pueden ser las canciones nos deberíamos enfrentar a otro tipo de problemas asociados a la naturaleza de este concepto. Un ejemplo podría ser el manejo de la palabra “feat” (o variaciones como “ft.” o “featuring”). Cuando “feat” aparece dentro del título de una canción, generalmente, significa que en dicho título se ha incluido el nombre de un artista colaborador. Por ello, tanto “feat” como las palabras que sigan son candidatos a ser descartados del nombre de la canción. No obstante, probablemente serían datos que se podrían usar con otros propósitos a pesar de no ser considerados parte esencial del título. En conjuntos de datos ruidosos o generados de forma manual es también posible encontrar palabras extra al principio o al final del título de una canción, haciendo referencia por ejemplo a un canal de radio en la que se emitió o, en caso de aparecer la palabra “live”, al lugar o evento de una representación en directo. La presencia de datos extra en el nombre de un trabajo musical puede resultar muy significativa en bases de datos especialmente ruidosas como aquellas formadas por la

unión de metadatos de archivos de audio independientes de origen diverso, como las formadas por software reproductor de música. Si la información se extrae de elementos etiquetados de forma errónea, es posible que nos encontremos títulos que, de hecho, contengan varios o incluso todos los campos de información (artista, fecha, género,...).

Otro ejemplo de entidad musical problemática a la hora de ser reconciliada por circunstancias específicas de su naturaleza son los géneros. Cuando manejamos géneros podemos encontrarnos con que una misma canción es etiquetada como "rock" en una fuente, como "pop" en otra y como "pop-rock" en una tercera. A veces incluso está confusión no se produce a la hora de asociar un género a un trabajo concreto, si no en la propia forma de denominar al género. Es decir, en ocasiones se usan distintas formas para referirse a una misma idea de estilo musical.

Nuestro convencimiento es que encontrar reglas de reconciliación entre dos bases de datos concretas relacionadas con el mundo de la música está lejos de ser una tarea trivial, de la misma forma que tampoco lo es encontrar reglas de reconciliación a aplicar para alguno de los campos concretos de tales bases de datos. El resultado puede verse notablemente afectado si tratamos de usar las mismas reglas de reconciliación empleadas con éxito para cierto par de fuentes si tratamos de darles carácter general y usarlas con cualquier par de fuentes y tipos de campos. La inferencia de reglas de reconciliación para pares de fuentes particulares puede llevarse a cabo mediante técnicas de aprendizaje de máquina a través de proporcionar datos de entrenamiento elaborados de forma manual [76]. No obstante, esta técnica puede resultar útil para cubrir cierto rango de problemas tales como las erratas o errores ortográficos, la detección de convenciones de nombres o la presencia de prefijos/sufijos en las cadenas a reconciliar, pero no son suficientes para la detección de casos en que la correspondencia de entradas no se puede deducir a través de patrones, como alias o los nombres artísticos frente a nombres civiles.

La arquitectura que hemos diseñado trata de adaptarse a estos escenarios en los que una misma realidad puede ser referida con múltiples formas y además puede estar relacionada con otras entidades mediante el uso de conceptos de grafos y tecnologías de web semántica. Nuestra propuesta consiste en representar toda la información de una de las fuentes a reconciliar en forma de grafo rdf. Mientras, las entradas de la otra base son utilizadas para elaborar consultas complejas que se lanzarán contra dicho grafo. El resultado de cada consulta sería una lista de los nodos que representan ciertas entidades que han sido detectados como más similares a los contenidos de la

consulta lanzada, de acuerdo a los siguientes criterios:

- Funciones de similitud entre cadenas de texto.
- Detección de formas representativas alternativas de un mismo concepto.
- Detección de entidades asociadas a un concepto con propósito desambiguatorio.

Para una mejora de resultados, MERA contempla enriquecer el grafo usado con fuentes de datos extra. Si tratamos de conciliar dos fuentes de datos A y B donde A contiene nombres civiles de artistas y B nombres artísticos, pero además contamos con una tercera fuente C en la que un artista es asociado con ambos tipos de nombre, entonces sería posible transformar la información de B en un grafo G y tratar de encontrar vínculos entre las entradas de C y G. Los resultados obtenidos pueden ser usados para añadir conocimiento en forma de tripletas extra a G, obteniendo un grafo G' que potencialmente puede contener artistas asociados tanto a su nombre artístico como a su nombre civil. Si en este punto usamos los datos de A para la elaboración de consultas contra G' podríamos obtener mejores resultados que si hubiésemos lanzado dichas consultas contra G.

MERA permite al usuario configurar qué conjunto de algoritmos deben ser aplicados a cada par de fuentes a reconciliar, o incluso qué algoritmos deben aplicarse sobre cada campo de cada una de esas fuentes, tratando así de adaptarse a los distintos escenarios mediante el uso del conocimiento del usuario sobre la naturaleza de los datos. Además, MERA permite la configuración de umbrales de similitud a aplicar en cada caso, de diferentes propiedades a tener en cuenta o de una serie de parámetros en la que se profundizará a lo lago de este trabajo. La arquitectura también contempla la posibilidad de que el usuario pueda implementar sus propias funciones de reconciliación y permite integrarlas en el flujo de ejecución del sistema.

5.1. Finalidad y financiación de proyecto

El presenta trabajo de investigación se financia a través de un acuerdo entre la empresa BMAT Licensing, S. L. y el grupo de investigación Weso Research Group, en el marco del Subprograma Avanza I+D del Ministerio de Industria, Turismo y Comercio. Los detalles del contrato son los siguientes:

- **Referencia**: TSI-020602-2012-195
- **Título del proyecto**: TagFlow, Sistema Avanzado de Monitoreo y Tagging de Flujos de datos multimedia
- **Entidad financiadora**: Ministerio de Industria, Turismo y Comercio, Subprograma Avanza I+D
- **Entidades participantes**: BMAT Licensing, S. L., Weso Research Group
- **Duración**: desde: 2012 hasta: 2014
- **Participación**: Investigador contratado
- **Investigador responsable**: BMAT Licensing S.L.

El presente trabajo tiene por finalidad, por un lado, la elaboración de un artículo de investigación candidato a ser publicado en una revista JCR. Por otro, la consecución de un sistema válido para los intereses empresariales de la compañía BMAT Licensing, S. L. El contenido presentado en este trabajo se combina con el trabajo del estudiante de máster Bernardo Martínez Garrido, encargado de la realización de una aplicación web que haga uso del prototipo de conciliación implementado.

A pesar de que la duración del contrato prevista en el plan avanza era de 2012 a 2014, nuestro grupo de investigación sigue en contacto con la empresa BMAT y continuamos trabajando en la misma línea. El trabajo que estamos presentando es aún un proyecto vivo y con modificaciones continuas, previstas al menos hasta finalizar el mes de julio de 2015, momento en el que ambas partes evaluarán la posible continuidad del proyecto.

5.2. Fijación de objetivos

Los objetivos del proyecto que presentamos son los siguientes:

- **Objetivos de la investigación**:
  - Exploración de técnicas de reconocimiento de entidades.
  - Exploración de características de datos relacionados con el mundo de la música.

  - Propuesta de mejora en la reconciliación de entidades relacionadas con el mundo de la música respecto a sistemas y técnicas existentes.
  - Aprovechamiento de tecnologías de web semántica.
  - Elaboración de un artículo de investigación para ser publicado en una revista JCR.

- **Objetivos contractuales**:

  - Elaboración de un sistema de reconciliación de entidades altamente configurable.
  - Integración del sistema que permita su integración con una aplicación web.
  - Investigación progresiva para la mejora de resultados de acuerdo a los requisitos del cliente.

5.3. Ámbitos de aplicación

El trabajo presentado está pensado para ser aprovechado por la empresa BMAT Licensing, S. L. para sus necesidad internas de reconciliación de entidades musicales. No obstante, la investigación se ha realizado con la idea de poder ser reutilizada en otros contextos.

El diseño ha sido planteado para ajustarse a los características y particularidades asociadas a las bases que contienen metadatos musicales. No obstante, se plantea como trabajo futuro el probar la eficacia de nuestros planteamientos en escenarios de reconciliación de entidades en que se manejen contenidos de otra naturaleza.

Capítulo 6

Estado del arte

Definición. La reconciliación o linkado de entidades, conocida también en la literatura como identificación de objetos (object identification) [69, 70], limpieza de datos (datacleaning) [19], matcheo aproximado o unión aproximada (aproximate matching or aproximate join) [28,29], matcheo vago (fuzzy matching) [2] o resolución de entidades (entity resolution) [6], consiste en la identificación de piezas de información referidas al mismo objeto del mundo real cuando no es posible disponer de identificadores únicos y confiables. Un uso básico de la reconciliación de entidades sería la detección de duplicados dentro de una base de datos o archivo o el descubrimiento de entradas referidas a la misma realidad en dos bases de datos diferentes.

La investigación realizada sobre reconciliación de entidades ha sido principalmente fundamentada sobre el trabajo de Fellegi and Sunter en 1969 [25], inspirado en las ideas introducidas por Newcombe [54]. La reconciliación de entidades es presentada como un problema de clasificación, en el que cada par de elementos enfrentados puede clasificarse como "matching" (coincidente) o "no matching" (no coincidente). Fellegi and Sunter proponen el uso de métodos no supervisados para esta tarea, estableciendo cuidadosamente la fundamentación probabilística de la teoría del linkado de entidades. Desde la publicación de su trabajo, han sido varias las comunidades científicas que han abordado este problema formulándolo desde su propio punto de vista, produciendo numerosas técnicas y tecnologías reutilizables para la resolución de entidades citewinkler2006overview, christen2012data.

Comparadores de cadenas. Puesto que la resolución de entidades es un problema debido a la ausencia de identificadores únicos confiables, las pro-

puestas tradicionales para encarar este problema están en su mayor parte basadas en el uso de comparadores de similitud en cadenas de texto sobre campos semi-identificativos [13, 77]. Al abordar un problema de linkado, es frecuente no encontrar situaciones de coincidencia exacta (carácter por caracter) entre campos semi-identificativos. Esto puede ser debido entre otras cosas a erratas o distintas convenciones de nombres/formato. De hecho, el reconocimiento de cadenas de texto no idénticas como representantes de la misma realidad es y ha sido desde hace años un campo de investigación muy activo en la ciencia computacional [13,30,51]. Se han realizado numerosos trabajos describiendo medidas de similitud entre cadenas de texto [11, 14, 49, 51] así como de métodos de linkado de individuos usando dichas medidas [5, 36, 37, 45, 75]. De forma particular, se ha llevado a cabo trabajo específico para el entorno de las bases de datos basándose en el uso de conocimiento previo sobre la naturaleza concreta de los datos tratados [32, 69, 70]. Los algoritmos de codificación fonética también han sido usados como medidas de similitud entre cadenas. La mayor parte del trabajo ha sido llevado a cabo para texto escrito en inglés [34, 42, 47], pero existen adaptaciones para otros idiomas [56].

Datos de entrenamiento. Decidir cual es la combinación de algoritmos más precisa a aplicar a la hora de enfrentarse a un escenario de reconciliación de entidades no es una tarea trivial [9]. Existen varios trabajos de investigación que demuestran que no existe un algoritmo o una combinación de los mismos que puedan mejorar la precisión y eficiencia de todos los demás en los distintos casos de aplicación de resolución de entidades [1, 14, 38, 77]. Esto es cierto incluso si nos centramos en ramas de algoritmos muy concretas tales como las distancias de edición entre cadenas de texto [55]. Varias líneas de investigación se han centrado en la obtención de métodos que permitan deducir de forma automática qué algoritmos o combinación de los mismos de entre un conjunto de posibilidades conocidas se adaptan mejor a la naturaleza de cada escenario particular. Estas técnicas deben recibir como entrada *datos de entrenamiento*, es decir, un conjunto de pares de entidades etiquetadas como "matching" o "no matching" por acción humana. Varias investigaciones han demostrado como el aprendizaje automático de máquina a partir de los datos de entrenamiento proporcionados por el usuario puede ser aplicados en resolución de entidades obteniendo una mejora de resultados, incluso si se trata de una pequeña cantidad de datos de entrenamiento [61, 76]. Las primeras aproximaciones hicieron uso de Modelos Ocultos de Markov [57] para

la deducción de parámetros [8, 9] siendo esta una técnica aún hoy extendida. No obstante, diferentes variantes o técnicas han sido propuestas para esta tarea [26], siendo ejemplos significativos las Cadenas Markov de Monte Carlo o los Campos Condicionales Aleatorios [17, 27, 44, 48].

Matching colectivo. Las técnicas utilizadas para determinar la similitud entre dos entradas a través del uso de métricas de comparación entre cadenas de texto aplicadas sobre sus campos son conocidas como *FBS*[1]. Existen no obstante ocasiones en que los algoritmos FBS no son suficientes para determinar de forma adecuada cuando dos entidades deben o no hacer matching, especialmente en casos en los que es necesaria un desambiguación [40]. En dichos escenarios, considerar un factor más la relación existente entre las entidades además de las propias características de cada entidad puede ser un mecanismo para la mejora de resultados.

Las técnicas tradicionales basadas en FBS realizan el linkado de cada individuo de forma independiente. Por contra, las técnicas en las que las relaciones entre entidades son tenidas en cuenta para producir un resultado son conocidos como técnicas de *matching colectivo* [7]. La representación de relaciones entre entidades encaja bien en las estructuras de datos basadas en grafos, por lo que este tipo de sistemas se basan generalmente en la representación de información mediante grafos [13]. Diferentes ideas han sido desarrollas, tales como la construcción de grafos de relaciones en los que las entidades se representan mediante nodos y las relaciones (posiblemente ponderadas de alguna forma) se representan con aristas entre ellos [7, 40], o el uso de grafos de dependencias, en los que un nodo representa la similitud entre un par de entidades y conecta con otros nodos mediante aristas si dicha similitud depende de otra similitud entre otro par [20]. Las aproximaciones de resolución colectiva combinadas con FBS pueden mejorar la calidad de los resultados de planteamientos basados puramente en FBS [7, 20, 40]. No obstante, los planteamientos de reconciliación colectivos suelen perder escalabilidad frente a los sistemas FBS [59], aunque ha sido demostrado que existen formas de adaptar dichos planteamientos mediante paralelización de tareas para resultar más escalables [59].

Técnicas de bloqueo. Los sistemas de linkado de entidades conllevan gestionar más tipos de desafíos que la pura detección de equivalencias entre

[1]Feature-Based Similarities

entradas, tales como la eficiencia o la escalabilidad cuando se manejan grandes volúmenes de datos. Si intentamos encontrar entidades en común entre dos conjuntos de datos $A = \{a_0, a_1, \ldots, a_n\}$ y $B = \{b_0, b_1, \ldots, b_m\}$, es muy deseable no tener que hacer una comparación entre cada entidad $a_i \in A$ y cada entidad $b_j \in B$. Esto es algo de vital importancia cuando los conjuntos de datos con los que se trata son, por ejemplo, bases de datos comerciales con millones o billones de entradas. La detección de una lista previa de potenciales parejas candidatas para reducir el número de comparaciones necesarias se conoce como *bloqueo*. Las primeras aproximaciones significativas fueron llevadas a cabo para el entorno de las bases de datos con datos sobre personas [53]. La técnica de bloqueo escogida fue considerar únicamente como pares potencialmente linkables aquellas entradas que tuviesen cierto(s) campo(s) en común, como los apellidos o la fecha de nacimiento. Desde la aparición de dicha propuesta, muchos planteamientos han sido probados, tales como la ordenación de entidades para mantener próximos aquellos contenidos que presentan más similitud [32], clustering de candidatos mediante funciones de menor complejidad que las técnicas de comparación a aplicar [11] o la indexación de q-gramas [4].

Sistemas existentes. Diseñar o desarrollar un sistema de reconciliación puede requerir la inclusión de todas o la mayor parte de las técnicas revisadas hasta ahora: algoritmos de comparación de cadenas, deducción de parámetros a través del uso de datos de entrenamiento, funciones de bloqueo, estrategias de matching colectivo, etc. Además se han de afrontar una serie de retos o decisiones extra tales como los distintos formatos de entrada/salida manejados, mecanismos de interacción con el usuario (API[2], librería de programación, aplicación web,...), opciones de configuración o la posibilidad de que el usuario incluye más algoritmos o estrategias de las que el paquete software ofrece originalmente.

Resulta complicado realizar una revisión de sistemas comerciales existentes, principalmente por dos razones. Primero, no es sencillo juzgar cuando un sistema es realmente *un sistema de reconciliación* y cuando la parte de reconciliación es sólo un modulo o una parte de un producto con otros objetivos. Segundo, porque dichos sistemas raramente publican sus detalles técnicos [13].

Entre los sistemas de reconocimiento de entidades open source o aque-

[2]Application Programming Interface

llos prototipos publicados para propósitos de investigación académicos, podemos encontrar varios paquetes de propósito general (no especializados en la reconciliación de metadatos relacionados con el mundo musical). Algunos ejemplos significativos de este tipo de aplicaciones serían Dude [21], D-Dupe [10, 41], SILK [72], BigMatch [78], FEBRL [12], FRIL [39], Merge ToolBox [63, 64], OYSTER [68], RRecordLinkage [62], WHIRL [15], NADEEF/ER [24] o PPRIL [43].

Todos estos sistemas has probado ser ser efectivos enfrentando cierto rango de problemas de resolución de entidades. No obstante, se trata de planteamiento heterogéneos que pueden ser clasificados desde varios puntos de vista:

- Capacidades principales: detección de duplicados en un mismo dataset, reconocimiento de entidades equivalentes en fuentes diferentes o una combinación de ambas.

- Interacción con el usuario: aplicación de interfaz amigable, API de programación, paquete de funciones...

- Propósito: resolución de problemas de reconocimiento de entidades, ser un benchmark de algoritmos de matcheado,...

- Grado de flexibilidad: opciones a la hora de escoger técnicas, algoritmos o estrategias a aplicar durante el proceso de matcheo, posibilidad para incluir nuevos módulos o funcionalidades al paquete original,...

- Disponibilidad de aprendizaje automático para la deducción de parámetros a través de datos de entrenamiento.

- Planteamiento puramente FBS o uso de reconciliación colectiva.

- Disponibilidad de funciones de bloqueo.

- Disponibilidad de otras características no mencionadas tales como la *reconciliación con con preservación de confidencialidad*, conveniente cuando los datos a reconciliar provienen de distintas entidades y tienen caracter sensible, o bien cuando tales datos han de ser tratados con encriptación [71].

Respecto a planteamientos especializados en la reconciliación de entidades musicales, hemos explorada patentes comerciales publicadas recientemente encontrando algunos resultados [22, 23]. No obstante, los trabajos encontrados son difícilmente comparables a MERA. Uno de ellos se trata de un sistema que permite al usuario conocer tantos metadatos de un CD introducido en sus sistema como sea posible [23], introduciendo en este proceso varios conceptos fuera de nuestro ámbito, tales como el manejo de varias bases de datos compuestas por unión de datos introducidos por distintos usuarios de la aplicación, el manejo de vectores de duraciones de pistas como representación de contenido de un CD o la reconciliación de elementos a través del uso de huellas de audio. Sin embargo, y a pesar del hecho de que el uso de fuentes de terceros para la mejora de los resultados del matching no es una idea reciente [73], esta patente es la única entre los sistemas explorados que incluye específicamente en su diseño el manejo, incluso con ponderación por fiabilidad, de varias fuentes de datos a la vez. El otro trabajo [22] propone una técnica basada en q-gramas que puede ser potencialmente aplicada en diferentes campos, pero cuya efectividad ha sido testeada contra bases de datos conteniendo entidades relacionadas con el mundo de la música. No obstante, este trabajo describe una estrategia más que un sistema, por lo que sería incluso factible incorporarla pro completo en el flujo de ejecución de MERA.

Un ejemplo significativo de sistema open source especializado en la integración de datos musicales es MusicBrainz Picard [66]. No obstante, este sistema, al igual que una de las patentes exploradas [23], usa el reconocimiento de entidades como parte de un sistema mayor para el etiquetado de metadatos de un CD. Ademas, MusicBrainz Picard está especialmente diseñado para reconciliar datos contra la propia fuente de datos de MusicBrainz, no para enfrentar dos pares cualquieras de fuentes.

En resumen, existen varios sistemas de reconciliación válidos, tanto especializados en entidades musicales como pensados para contenidos de cualquier tipo de naturaleza, pero ninguno de ellos propone algunos de los puntos incluidos en la especificación de MERA:

- Conciliación colectiva de entidades usando un esquema de grafo en el que se refleja la procedencia (las fuentes originales) de cada pieza de información.
- Estrategia de navegación de grafos adaptada al esquema anterior en la que, dado cierto nodo n_α que representa una entidad, se explora su

región adyacente en el grafo en busca de distintas formas de representación de la propia entidad o bien otras entidades relacionadas.

Capítulo 7

Descripción de la arquitectura MERA

MERA define una estrategia para combinar varias categorías de algoritmos o heurísticos con el propósito de conciliar dos conjuntos de entidades $A = \{a_0, a_1 \dots a_n\}$ y $B = \{b_0, b_1 \dots b_m\}$, donde las entidades de B son representadas mediante un grafo rdf relacional G y las de A son utilizadas para construir consultas que se lanzarán contra G. Cada elemento $a_i \in A$ es procesado de forma independiente para encontrar aquellos elementos pertenecientes a B que más se le asemejan. El usuario de MERA debe tener la posibilidad de definir un conjunto de normas que determinen las condiciones que una entidad $b_j \in B$ debe cumplir para ser considerada como un posible resultado. Consecuentemente, cada entidad $a_i \in A$ puede ser asociada a cero, uno o varios resultados dependiendo del número de entidades b_j que encajen en las condiciones configuradas. En el caso de que una consulta arroje varios resultados para una misma entidad (consulta) a_i, dichos resultados serán ordenados y presentados en orden descendente de grado de similitud calculado.

Si pensamos en MERA como una caja negra, podríamos definir sus entradas y salidas de la siguiente forma:

- *Entrada1*: conjunto de entidades $A = \{a_0, a_1 \dots a_n\}$ del cuál extraer consultas.
- *Entrada2*: grafo rdf representando las entidades de un conjunto $B = \{b_0, b_1 \dots b_m\}$, así como las distintas relaciones que puedan existir entra las mismas.

- *Entrada3*: especificación de aspectos configurables de MERA. Tales aspectos se describirán en profundidad en la sección 7.4.

- *Salida*: asociación de cada elemento $a_i \in A$ con aquellos elementos $b_j \in B$ que encajen en las condiciones de reconciliación definidas.

Las posibles configuraciones recibidas en la *Entrada3* incluyen, entre otros aspectos, algoritmos de reconciliación a aplicar sobre cada tipo de nodo, limite máximo de entidades de B a devolver en cada consulta o umbral mínimo de similitud a alcanzar tras la aplicación de los algoritmos para que una entidad $b_j \in B$ sea candidata a ser incluida en los resultados. En la sección 7.4 se profundizará en las posibles configuraciones.

Uso de fuentes de datos extra. A pesar del hecho de que hemos descrito MERA como una arquitectura diseñada para la reconciliación de dos fuentes, más datastes pueden ser involucrados en el proceso para la mejora de resultados. Digamos que contamos con un conjunto de datos A conteniendo artistas identificados por su nombre civil y un conjunto de datos B que también contiene artistas pero almacenados por nombre artístico. Si tratásemos de reconciliar los elementos de A y B, con independencia de los algoritmos escogidos, probablemente sólo tuviésemos éxitos en casos en los que el nombre artístico y civil de la persona representada coinciden (ejemplo: "Jennifer López") o cuando, por lo menos, comparten algunos tokens (ejemplo: "Shakira" vs "Shakira Isabel Mebarack Ripoll"). Sin embargo, para aquellas personas cuyos nombres artístico y civil no comparten secuencias significativas de tokens/caracteres (ejemplo: "Stefani Joanne Angelina Germanotta" vs "Lady Gaga"), no sería posible detectar matches.

Para estas situaciones, proponemos el uso de fuentes de datos extra para enriquecer los contenidos originales de los conjuntos A y B. Supongamos que contamos con una tercera fuente C en la que para un misma artista se recogen tanto su nombre artístico como su nombre civil. La fuente C podría ser usada como puente para la reconciliación de A y B por el uso de reglas transitivas. Siendo a_α une entidad en A, b_α una entidad en B ,c_α una entidad en C y representado un link entre dos entidades u y v como $u \Leftrightarrow v$, si $a_\alpha \Leftrightarrow c_\alpha$ y $b_\alpha \Leftrightarrow c_\alpha$, entonces $a_\alpha \Leftrightarrow b_\alpha$.

La forma de hacer uso de las fuentes extra para el enriquecimiento de datos sería la utilización previa de MERA para descubrir links entre B, representada por un grafo G, y las nuevas fuentes. El conocimiento obtenido

con los resultados se añadiría a G obteniendo un grafo enriquecido G'. Finalmente, se lanzarían las consultas construidas con los datos de entidades de A contra G'. En el ejemplo anterior, el grafo resultante contendría las entidades originales de B con sus nombres artísticos intactos más los respectivos nombres civiles de aquellas entidades de B que hayan hecho link con contenido de C.

Existen bases de datos de acceso público con grandes cantidades de datos de entidades relacionadas con la música que pueden ser usadas en para este propósito, tales como Discogs [31], Allmusic [50] y MusicBrainz [67].

Estructura y construcción de consultas. Al reconciliar dos fuentes de datos A y B con MERA, los datos contenidos en B son representados mediante un grafo relacional G, mientras que los datos de A son usados para la construcción de consultas para lanzar contra el G. Dichas consultas pueden involucrar, y es conveniente que involucren, información de varias entidades de forma simultánea. Una consulta en el contexto de MERA está formada por un **tipo principal** o, un **contenido principal** m y un conjunto R con un número indeterminado de **refinamientos** o información de desambiguación. Cada refinamiento consiste en un par $r_i = (t_i, s_i)$, donde s_i es una cadena de texto que representa algún concepto o entidad y t_i es su tipo.

Consideremos un escenario en el que tenemos un conjunto de datos A conteniendo artistas que pueden estar asociados a canciones. Uno de los artistas de A es A_α =“Mike Chang”, que no tiene ninguna canción asociada. Otro artista sería A_β =“Kurt Hummel”, con la canción asociada A_γ =“For Good”. Ahora supongamos que queremos reconciliar todos los artistas presentes en A con los contenidos de un grafo G. Para ello, hemos de usar las entidades de A para construir consultas. La consulta resultante $q_\alpha = (o_\alpha, m_\alpha, R_\alpha = \{r_\alpha *\})$ para A_α contendría o_α = “artista”, m_α =“Mike Chang” y $r_\alpha = \{\lambda\}$, siendo $\{\lambda\}$ un conjunto vacío. La consulta resultante $q_\beta = (o_\beta, m_\beta, R_\beta = \{r_\beta *\})$ para A_β contendría o_β =“artista”, m_β =“Kurt Hummel” y R_β = {(“canción”,“For Good”)}.

Si A_β tuviese más canciones asociadas, más pares tipo r_{β_i} = (“canción”, s_{β_i}) serían añadidos a R_β. Si conociésemos algún otro dato como por ejemplo que Kurt Hummel ha nacido en USA, podríamos añadir a los refinamientos R_β un refinamiento tal que (“país”,“USA”).

Computación paralela. En el proceso de reconciliación de las fuentes A y B, cada consulta generada a partir de los datos de A se procesa de forma independiente de las demás consultas. Además, el grafo G que representa los datos del conjunto B es usado en modo solo-lectura durante el proceso de linkado. En este escenario, pensamos que la inclusión de mecanismos de computación paralela mediante la ejecución de consultas en hilos/nodos máquina independientes puede ser factible.

En cuanto a la capa de datos en una implementación de MERA con una estrategia de computación paralela de consultas como la que hemos descrito, se debería contar con un sistema que contenga el grafo G capaz de atender a varias peticiones simultáneas de forma asíncrona para atender a los diferentes hilos de consulta sin formar un cuello de botella. No obstante, la replicación de datos total o parcial de G en copias sin sincronización entre ellas sería suficiente puesto que el contenido de G no va a ser mutado durante la ejecución del proceso de reconciliación. Por la misma razón, sería posible el uso de entrada/salida no bloqueante en la interacción con G, puesto que los datos serán accedidos en modo de solo-lectura.

7.1. Algoritmo adaptable de MERA

Ejecución de consultas. En el algoritmo 1, hemos descrito en pseudocódigo la forma de ejecutar de MERA desde el momento en que recibe un consulta (query) hasta que se devuelven todos los nodos de un grafo G con mayor grado de similitud encontrados. No obstante, algunos métodos, funciones y propiedades usados en dicho algoritmos necesitan cierta explicación para la comprensión completa del algoritmo:

- Propiedad *nodes*: invocada sobre un objeto Graph (Grafo) G, devuelve un conjunto que contiene todos los nodos de G.
- Propiedad *blockingFunctions*: invocada sobre un objeto Config (Configuración), devuelve un set que contiene funciones de bloqueo definidas por el usuario.
- Funciones f_{BLK}. Cada función f_{BLK} espera recibir una Query (consulta) q y un conjunto de nodos S, y devuelve un subconjunto de nodos $S' \in S$ que contiene los mejores candidatos a hacer matching con q en S de acuerdo al criterio de bloqueo de f_{BLK}.

Algorithm 1 MERA Query

Require: Graph G
Require: Query q
Require: Config C
candidateNodes $\leftarrow G$.nodes
for all $f_{BLK} \in$ C.blockingFunctions **do**
    candidateNodes $\leftarrow f_{BLK}(q,$ candidateNodes $)$
tmpResults $\leftarrow [\lambda]$
for all aNode $\in$ candidateNodes **do**
    formScores $\leftarrow \{\lambda\}$
    for all aForm $\in$ aNode.alternativeForms **do**
        formScores $\leftarrow$ formScores $+ f_{CMP}(q$.mainContent, aForm , q.type)
    if máx(formScores) $\geq C$.minMainScore(q.order) **then**
        tmpResults[aNode] = máx(formScores)
for all aRefType $\in q$.refinementTypes **do**
    for all aCandidateRes $\in$ tmpResults **do**
        candidateRefs $\leftarrow G$.getRelated(aCandidateRes.node, aRefType)
        for all aRefinement $\in q$.refinementsOfType(aRefType) **do**
            formScores $\leftarrow \{\lambda\}$
            for all aForm $\in$ candidateRefs.alternativeForms **do**
                formScores $\leftarrow$ formScores $+ f_{CMP}($ aRef, aForm, aRefType)
            if máx(formScores) $\geq C$.minRS(q.order, aRefType) **then**
                refScore $\leftarrow$ máx(formScores) $\cdot$ C.relev(q.order, aRefType)
                tmpResults[aNode] $\leftarrow$ tmpResults[aNode]+ refScore
results $\leftarrow [\lambda]$
$k \leftarrow C$.maxResults(q.order)
for all candidate $\in$ tmpResults **do**
    if candidate.score $< C$. minScore(q.order) **then**
        results$\leftarrow$ results + candidate
sort(results)
if |results| $\leq k$ **then**
    return results
else
    return results[0: k]

- Estructura $\leftarrow \{\lambda\}$: representa un conjunto vacío.
- Estructura $[\lambda]$: representa un diccionario vacío, en el que se almacenan pares de la forma (clave $\rightarrow$ valor).
- Propiedad *alternativeForms*: devuelve todas las cadenas de texto que representan formas alternativas de aquel nodo n sobre el que es invocada.
- Función f_{CMP}: espera dos cadenas de texto s y t y un el tipo principal de una consulta (orden) o y devuelve un grado de similitud entre s y t en el rango $[0, 1]$ calculado con los algoritmos de comparación asociados al tipo o.
- Propiedad *mainContent*: devuelve la cadena de texto principal m de la Query q sobre la que es invocada.
- Propiedad *type*: devuelve el tipo principal o de la Query q sobre la que es invocada, es decir, el tipo de la cadena de texto principal m en la Query q.
- Método *minMainScore*: invocado sobre un objeto Config C recibe el tipo principal o de un objeto Query y devuelve el grado de similitud mínimo que una cadena de texto debe alcanzar al compararse con otra cadena de tipo o para ser tenida en cuenta en los resultados.
- Propiedad *order*: contiene el tipo principal del objeto Query q sobre el que es invocada.
- Propiedad *refinementTypes*: Contiene un conjunto con todos los tipos de los refinamientos incluidos en el objeto Query q sobre el que es invocada.
- Método *getRelated*: invocado sobre un objeto Graph G, espera recibir un nodo n y el tipo de un t refinamiento. Devuelve todos los nodos relacionados con n a través de una arista etiquetada como de tipo t.
- Método *refinementsOfType*: invocado sobre un objeto Query q, espera recibir un tipo t de refinamiento y devuelve un conjunto con todos los cadenas de texto en refinamientos de q asociados al tipo t.

- Método *minRS*: invocado sobre un objeto Config C, espera un tipo principal de consulta o y un tipo de entidad t, y devuelve la puntuación mínima que una entidad de tipo t debe alcanzar para ser tenida en cuenta en los resultados al ser el refinamiento de una consulta cuyo tipo principal es o.

- Método *relev*: invocado sobre un objeto Config C, espera un tipo de principal de consulta o y un tipo de refinamiento t, y devuelve el factor de relevancia de un refinamiento con tipo t cuando se ejecuta una consulta cuyo tipo principal es o.

- Método *maxResults*: invocado sobre un objeto Config C, espera un tipo de principal de consulta o y devuelve el número máximo de resultados por consulta cuando se ejecuta una orden de tipo o.

- Función *sort*: espera un conjunto o un diccionario S y modifica la posición interna de los elementos dentro de S para colocarlos en orden descendente.

Hemos distinguido cuatro etapas principales en el algoritmo 1. Esta división nos ayuda a presentar el algoritmo de una manera menos formal como la sucesión de las siguientes cuatro etapas:

- *Etapa de bloqueo (líneas 1-4).* El objeto Config C almacena una lista de funciones de bloqueo para el filtrado de nodos candidatos en el objeto Graph G. Todas esas funciones son aplicadas secuencialmente sobre los nodos de G para obtener una lista de nodos candidatos a hacer matching con el objeto Query q.

- *Comparación principal (líneas 5-10).* El objeto Query q tiene un tipo principal o y un contenido principal m. m es comparado con todas las formas alternativas de las entidades identificadas por los nodos candidatos obtenidos en la etapa anterior.

- *Refinamientos (líneas 11-20).* El objeto Query q puede contener una serie de datos extra o *refinamientos*. Por cada refinamiento en q, G es navegado tomando como punto de partida los nodos candidatos. Si durante la exploración de G se encuentran entidades relacionadas con el refinamiento de puntuación lo bastante alta de acuerdo a los criterios fijados en C, se añade cierta puntuación calculada a partir del grado

de similitud encontrado y de factores de ponderación definidos en C al nodo n desde el cual empezamos a navegar el grafo.

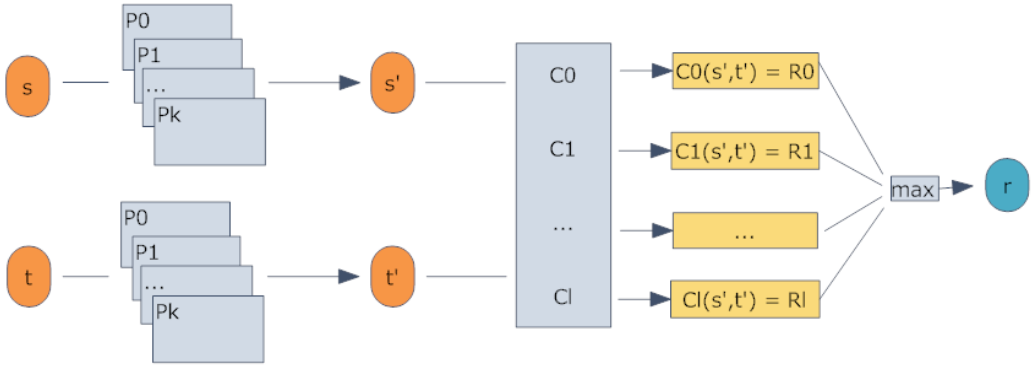
- *Filtrado, devolución y ordenación de resultados (líneas 21-30).* Los resultados asociados a un objeto Query q son colocados en orden descendente de acuerdo a su puntuación de similitud obtenida. El usuario puede definir una cantidad de resultados máxima k a devolver asociados a una query q en caso de que haya suficientes nodos que cumplan los mínimas condiciones definidas en C. Si se encontrase una cantidad de resultados mayor que k sólo los k mejores serían retornados. De otro modo, se devolverían todos los resultados.

Comparación de cadenas de texto En las líneas 8 y 17 del algoritmo 1 usamos la macro f_{CMP} para representar el proceso de cálculo de similitud entre dos cadenas de texto s y t a través del uso de varios algoritmos de estandarización y comparación de cadenas. Además de s y t, la macro f_{CMP} también espera recibir como parámetro el tipo de datos o al que pertenecen las entidades identificadas con s y t. o debe ser proporcionado a la función para seleccionar qué conjunto de algoritmos es el adecuado según la configuración del usuario para detectar la similitud entre s y t.

Para justificar que los distintos tipos de datos puedan necesitar diferentes conjuntos de algoritmos, supongamos que queremos encontrar links entre dos conjuntos de datos A y B, conteniendo ambos entradas sobre nombres de canciones y fechas de lanzamiento de las mismas. Un intento de estandarizar un nombre de canción podría requerir la eliminación de espacios en blanco redundantes, paso a minúsculas de todos los caracteres, etc. Sin embargo, la estandarización de una cadena de texto que representa una fecha probablemente requiera de la detección del formato de fecha usado en cada caso y de la transformación si fuese necesario de s y t a un formato común. Para las tareas de comparación, en el caso de los nombres de canciones una táctica potencialmente válida sería el uso de algoritmos de detección de distancia de edición entre cadenas de texto, mientras que para el caso de las fechas es probable que las entradas se deban parsear y transformar para devolver una distancia en algún tipo de unidad de tiempo entre s y t.

Al mapear dos cadenas de texto s y t a un grado de similitud r, una vez el conjunto de funciones de pre-procesamiento (estandarización) $P = \{P_0, P_1, \ldots, P_k\}$ y el conjunto de funciones de comparación de cadenas $C = \{C_0, C_1, \ldots, C_l\}$ a aplicar ha sido seleccionado en base al tipo de datos o,

Figura 7.1: Mapeo de dos cadenas s y t a un número real r



MERA sigue siempre el mismo flujo de acciones para llegar a un resultado. Dicho flujo es descrito gráficamente en la figura 7.1. s es transformado en s' y t en t' mediante el encadenamiento de todas las funciones $P_i \in P$. Esto es, la entrada de P_0 sería s y, en términos generales, la entrada de P_i sería la salida de P_{i-1}. Los resultados s' y t' serían la salida de P_{i-1}. Tras ello, para cada $C_i \in C$, calcularíamos cada $R_i = C_i(s', t')$. El valor más grande de $\{R_0, R_1, \ldots, R_i\}$ sería usado como resultado r, teniendo $f_{CMP}(s, t, o) = r$.

7.2. Métodos

MERA define una estrategia para combinar tres categorías diferentes de funciones: estandarización o pre-procesamiento, bloqueo y comparación de cadenas. Cada conjunto de funciones será usado de forma conjunta para devolver un resultado sobre cada consulta, es decir, para detectar un conjunto de entidades similares representadas en un grafo G a cada entidad a_i perteneciente a un conjunto A. El usuario de una implementación de MERA debe poder definir qué conjuntos de funciones de cada categoría han de ser aplicadas al reconciliar cadenas de texto de distintos tipos. En las próximas secciones haremos un repaso de la naturaleza y las posibilidades de cada categoría, además de explicar cómo el sistema combina las funciones especificadas por el usuario y cómo se comporta con las distintas posibles configuraciones.

7.2.1. Estandarización o pre-procesamiento

Un problema típico a enfrentar cuando se comparan texto de diferentes fuentes (o incluso de una misma fuente en la que no se siguen patrones consistentes) es la necesidad de aplicar transformaciones o heurísticos para la estandarización de datos [33]. Nos referimos a diferencias entre cadenas potencialmente no significativas tales como uso de mayúsculas o minúsculas, presencia de espacios en blanco redundantes, presencia de marcas de puntuación o convenciones de nombres. No obstante, puede ser necesario incluir estrategias de estandarización de más aspectos, e incluso los ejemplos citados pueden ser malos ejemplos en determinados casos, o al menos no tan triviales como pueda parecer a simple vista. Por ejemplo, cuando se tratan acrónimos, la decisión de borrar, reemplazar o preservar marcas de puntuación puede tener bastante impacto sobre el resultado final. Por ejemplo, consideremos los acrónimos "B.O.B.", que representa al artista "Bobby Ray Simons" y "S.O.A.D." que representa al grupo "System Of A Down". Con respecto al tratamiento de los signos de puntuación, el efecto de conservarlos, reemplazarlos o borrarlos sería el siguiente:

- *Eliminar signos de puntuación*: las entradas serían transformadas en "SOAD" y "BOB". Si "SOAD" estuviese almacenado en el grafo como una forma alternativa del grupo "System Of A Down", estaríamos facilitando el proceso de reconciliación. Si "BOB" estuviese también almacenado como forma alternativa de "Bobby Ray Simons", entonces habremos obtenido un matching exacto entre la cadena de texto original y una de las formas del artista objetivo. No obstante, al mismo tiempo, hemos introducido la posibilidad de que aparezcan entre los resultados un elevado número de falsos positivos, debido a que al ser Bob un nombre propio común podríamos encontrarnos con que fuesen muchas otras las entidades que tuviesen registrado "Bob" como forma alternativa: Bob Dylan, Bob Marley, Bob Sinclair,... o Bob Esponja, como artista colaborador de canciones infantiles.

- *Conservar signos de puntuación*: Las entradas permanecerían sin variaciones. Para producir un match apropiado es probable que necesitásemos que las formas "B.O.B." y "S.O.A.D." estuviesen registradas en el grafo como formas alternativas de sus respectivas entidades. Si nos encontrásemos en un escenario en el que las formas registradas son

"SOAD" y "BOB", puede que no fuese suficiente para detectar similitud. Si los algoritmos de comparación aplicados fuesen capaces de ponderar la presencia de caracteres con mayor significancia que la presencia de marcas de puntuación sí se podría alcanzar un resultado apropiado, pero volveríamos a encontrarnos en una situación de potenciales falsos positivos semejante a la descrita en el caso anterior.

- *Sustitución de signos de puntuación por caracteres en blanco*: las formas resultantes serían "S O A D" y "B O B". Dependiendo de los algoritmos de comparación escogidos, puede resultar difícil la reconciliación de estas formas contra otras como "SOAD" o "BOB". No obstante, con respecto a otro tipo de entradas, nos estamos asegurando de que no introduciremos palabras equivocadas por hacer uso de heurísticos de estandarización equivocados. Imaginemos una fuente ruidosa en la que por alguna razón se proporcionan nombres de canciones con palabras separadas por puntos o barras bajas, tal como "Amor.de.verano". Eliminando signos de puntuación estaríamos deformando el título de la canción ("Amordeverano"); conservándolos, estaríamos manteniendo un ruido innecesario en la entrada; sustituyéndolos por espacios en blanco, en cambio, obtendríamos el título que probablemente más se asemeje al nombre canónico real de la canción referida ("Amor de verano").

Un ejemplo de estandarización más compleja que los presentados hasta ahora sería el intento de eliminar palabras potencialmente vacías de significado, tales como preposiciones, conjunciones o artículos. Esto puede ser una tarea relativamente sencilla cuando tratamos con fuentes que incluyen información en un único idioma, aunque, como en el caso anterior, con excepciones. Pongamos de ejemplo la canción en castellano "La, la, la", de "Massiel". En castellano la secuencia "la" sería detectada como un artículo, entrando en la categoría de palabras que potencialmente no aportan significado. Una estandarización sobre dicho nombre nos dejaría el título "„", o incluso una cadena vacía en caso de aplicar también eliminación de signos de puntuación. En escenarios en los que podemos recibir texto en distintos lenguajes el problema gana en complejidad, puesto que lo que para una lengua puede ser una palabra vacía puede ser para otra una secuencia con carga semántica. Un ejemplo sería el artículo "la" en castellano ya mencionado, que al aparecer en textos en inglés suele referirse al acrónimo de la ciudad de "Los Ángeles".

Diferentes técnicas han de ser seleccionadas teniendo en cuenta la naturaleza del contenido recibido, las convenciones de nombres usadas, la presencia o no de patrones consistentes y la calidad de los datos. Los conjuntos de algoritmos escogidos de otras categorías (comparación y bloqueo) también podrían representar un factor a tener en cuenta a la hora de escoger las técnicas de pre-procesamiento a utilizar.

Todos los algoritmos de estandarización seleccionados o incorporados por el usuario en una implementación de MERA deben de cumplir cierto interfaz de comportamiento: recibir una cadena de texto como único parámetro y retornar la cadena transformada de acuerdo a los criterios del algoritmo. La forma retornada será la usada en las fases de bloqueo y comparación. En caso de que el usuario no especifique ningún algoritmo de estandarización en la configuración de MERA, el comportamiento por defecto debería ser el uso de la forma original sin ningún tipo de transformación en el resto de fases.

7.2.2. Comparación de cadenas

Cuando tratamos de encontrar si existe una equivalencia entre dos entradas distintas en escenarios en los que no es posible hacer uso de identificadores únicos confiables, la mayor parte de la carga en el proceso de reconciliación recae en el uso de medidas de similitud entre cadenas de texto [13]. Por ejemplo, si tratamos con bases de datos personales de individuos reales, es probable que sea necesario aplicar este tipo de técnicas sobre campos semi-identificativos de cada entidad, somo pueden ser la combinación de nombre, apellidos y fecha de nacimiento, para detectar aquellos pares que potencialmente se están refiriendo a la misma persona real.

Podemos definir las funciones de similitud como funciones que mapean dos cadenas de texto s y t a un número real r, donde valores más altos de r equivalen a un mayor grado de similitud entre s y t [14]. La cifra resultante r es habitualmente un valor en el rango $[0, 1]$, donde 1 significa igualdad de acuerdo a los criterios del algoritmo y 0 significa que no se ha detectado ningún parecido entre las entradas s y t. Todos los algoritmos de similitud integrado en MERA han de cumplir dicho interfaz, es decir, han de esperar recibir dos cadenas de texto s y t y han de devolver número real r que expresa su grado de similitud cumpliéndose $r \in [0, 1]$.

Existen varios tipos de algoritmos pensados para medir el grado de similitud entre dos cadenas de texto. Todos ellos son potencialmente integrables en el flujo de ejecución de MERA. En las siguientes secciones haremos un

breve repaso de estas técnicas.

Basados en caracteres

Los algoritmos de comparación basados en caracteres, también llamados algoritmos de distancia de edición [51], tratan las cadenas de entrada s y t a comparar como una sucesión ordenada de caracteres. Generalmente basan sus resultados en el número de operaciones de borrado, adición, sustitución y, en algunos casos, trasposición necesarias para transformar la secuencia s en la secuencia t.

Un ejemplo significativo de este tipo de algoritmos sería la similitud de Levenshtein. El algoritmo de distancia Levenshtein [46] asigna un coste unitario a los tres tipos básicos de operaciones de edición: borrado, añadido y substitución de caracteres. Dadas dos cadenas s y t, la distancia de Levenshtein α entre ambas sería el número mínimo de operaciones de edición necesarias para transformar s en t. La distancia de Levenshtein se transforma en una función de similitud que devuelve α' donde $\alpha' \in [0, 1]$ a través de calcular el número máximo de operaciones de edición $\beta = \text{máx}(|s|, |t|)$ para transformar s en t, y devolviendo un resultado $r = \frac{\alpha}{\beta}$.

Una variación de la similitud de Levenshtein sería la similitud Damerau. La distancia de Damerau [18], comúnmente referida como Damerau-Levenshtein, añade una cuarta operación de edición: la transposición de caracteres. De esta forma, dadas las cadenas 'thing'y 'thnig', La distancia de Levenshtein entre ellas sería 2 (dos operaciones de sustitución), mientras que la distancia de Damerau sería 1 (una operación de trasposición).

La forma de transformar la distancia Damerau en una función de similitud es idéntica a la de Levenshtein. Siendo α el número mínimo de operaciones entre dos cadenas s y t y siendo $\beta = \text{máx}(|s|, |t|)$ el resultado r de la función de similitud de Damerau sería $r = \frac{\alpha}{\beta}$.

Otros ejemplos significativos de este tipo de estrategias de distancia de edición serían Jaro [36] y su extensión Jaro-Winkler [74]. Estudios recientes indican que estos dos últimos algoritmos mejoran la efectividad de otras técnicas similares cuando se reconcilian nombres de personas con datos conteniendo altas tases de errores [55].

Existe una subcategoría de funciones de similitud basadas en caracteres que afrontan las mediciones desde un punto de vista diferente: algoritmos de alineamiento. Algunos ejemplos significativos serían Smith-Waterman [65] and Needleman-Wunsch [52], ambos originalmente diseñados para la compa-

ración de secuencias biológicas tales como cadenas de amino-ácidos. Estos algoritmos tratan de hallar el mejor alineamiento de caracteres entre s y t. Esto es, transforman s y t en $s' = \{s'_0, s'_1, \ldots, s'_k\}$ y $t' = \{t'_0, t'_1, \ldots, t'_k\}$ donde hay tanto casos como sea posible en los que se da que $s'_i = t'_i$. Para transformar s en s', *caracteres nulos* o *gaps*, denotados como λ, pueden ser añadidos en cualquier posición de s y t, pero no contarán como una coincidencia entre s' y t' en caso de que se diese que $s'_i = t'_i = \lambda$. Needleman-Wunsch busca el mejor alineamiento global, es decir, la situación en la que hay más pares s'_i y t'_i coincidentes. Por contra, Smith-Waterman busca el mejor alineamiento local, es decir aquel alineamiento en el que haya el mayor número de casos consecutivos seguidos en los que $s'_i = t'_i$.

Basados en tokens

En contraste a lo propuesto por las funciones basadas en caracteres, las estrategias basadas en tokens tratan las cadenas de entrada s y t como un *multi-conjuntos* o *bolsas* de tokens (palabras), es decir, como conjuntos con repetición de tokens sin noción de orden entre ellos [14]. Estos algoritmos basan sus resultados en la presencia/ausencia de tokens compartidos en las cadenas s y t.

Un ejemplo simple de este tipo de funciones sería el índice Jaccard [35]. Considerando las cadenas s y t como conjuntos de palabras S y T respectivamente, la similitud de Jaccard Δ_{JAC} entre s y t sería $\Delta_{JAC}(s,t) = \frac{|S \cap T|}{|S \cup T|}$.

Otro ejemplo significativo sería el extendido TF-IDF o similitud de coseno [16]. Considerando las cadenas s y t como conjuntos de palabras S de longitud n y T de longitud m, la similitud de coseno Δ_{TFIDF} se define como sigue [16]:

$$\Delta_{TFIDF}(S,T) = \sum_{w \in S \cap T} V(w,S) \cdot V(w,T) \tag{7.1}$$

donde $TF_{w,S}$ es la frecuencia de aparición de la palabra w en el set S, IDF_w es el inverso de la fracción de nombres que contienen la palabra w en el conjunto de datos estudiados o *corpus*,

$$V'(w,S) = \log(TF_{w,S} + 1) \cdot \log(IDF_w) \tag{7.2}$$

y

$$V(w,S) = \frac{V'(w,S)}{\sqrt{\sum_{i=0}^{n} V'(w_i,S)^2}} \tag{7.3}$$

Presentar este último algoritmo como compatible con MERA puede resultar controvertido en el sentido de que se necesita información extra más allá del puro contenido de las cadenas s y t para alcanzar un resultado. Necesitamos conocer la frecuencia de aparición de cada uno de los tokens presentes en s y t en el corpus. Obviamente, esa información es accesible a través del contenido del grafo, pero si tratamos grandes volúmenes de datos sería computacionalmente prohibitivo navegar el grafo por completo para deducir la frecuencia de aparición de cada token cada vez que se deba realizar una comparación entre dos cadenas de texto. En lugar de eso, la solución natural sería poder consultar una estructura extra en la que ese tipo de información fuese accesible con una complejidad operacional admisible. No obstante, esta estructura debería ofrecer exactamente la misma información que la que se obtendría a través de navegar el grafo por completo en cada consulta. Esto es, dicha estructura extra debe estar totalmente sincronizada con el contenido del grafo.

La forma en la que esta estructura es mantenida y consultada esta fuera del ámbito de este trabajo, pero MERA está diseñada para admitir ese tipo de lógica mientras el algoritmo a usar respete el interfaz de llamadas establecido. De hecho, hemos implementado un prototipo que usa un tipo de función de bloqueo que requiere el uso de un índice de q-gramas sincronizado con el contenido en el grafo. Describiremos en que consiste dicha función en la sección 7.2.3.

Distancias híbridas

Algunas algoritmos basados en tokens funcionan apoyándose sobre otras funciones basadas en caracteres con el objetivo de paliar el efecto en el resultado de pequeñas variaciones en algunos tokens. Siendo Δ_{JAC} la función de similitud de Jaccard y Δ_{LEV} la función de similitud de Levenshtein, $s =$ "Shakira Isabel", $t =$ "Isabel Shakira", $t' =$ "Shakira Izabel" y $t'' =$ "Izabel Shakira", tendríamos lo siguiente:

- $\Delta_{JAC}(s,t) = 1$ y $\Delta_{LEV}(s,t) \approx 0$
- $\Delta_{JAC}(s,t') = 0{,}5$ y $\Delta_{LEV}(s,t') \approx 1$
- $\Delta_{JAC}(s,t'') = 0{,}5$ y $\Delta_{LEV}(s,t'') \approx 0$

Siendo Jaccard y Levenshtein funciones representativas de tokens y caracteres respectivamente, podemos ver a través de los anteriores ejemplos

que hay algunas situaciones en los que ninguna de las dos categorías de algoritmos puede detectar con efectividad un alto grado de similitud entre dos cadenas de texto cuando un humano podría fácilmente identificarlas como equivalentes. Nos referimos a la comparación entre "Shakira Isabel" y "Izabel Shakira", donde $\Delta_{JAC}(s, t'') = 0{,}5$ y $\Delta_{LEV}(s, t'') \approx 0$. Los algoritmos basados ven su resultado afectado de forma crítica cuando un error tipográfico es introducido en alguno de los tokens variando al menos un caracter, sobre todo cuando se trabaja con cadenas con poca cantidad de caracteres como es el caso de s o t''. Los algoritmos basados en caracteres, por contra, devuelve un resultado que puede resultar inadecuado cuando ha habido una alteración del orden de caracteres. Cuando ambas circunstancias se dan al mismo tiempo, como en el par (s, t''), ninguna de las funciones es capaz de detectar un match que en realidad sí existe.

Las funciones híbridas combinan ambos tipos de algoritmos para ser efectivas también en casos como el descrito en el párrafo anterior. Ejemplos sifnificativos de algoritmos de comparación híbridos serían el comparador Monge-Elkan [49] o la función Soft TF-IDF [14]

Heurísticos hechos a medida

Puede haber funciones que no encajen exactamente en ninguna de las categorías que hemos descrito pero que de igual forma pueden y deben encajar en el flujo de acciones de MERA para determinados escenarios. Dichas funciones podrían ser de propósito general o fuertemente ligadas a la naturaleza de los datos de un caso concreto. Supongamos por ejemplo que pretendemos descubrir link entre entradas de dos conjuntos de datos A y B, ambos conteniendo datos sobre nombres de artistas. Los artistas de A son almacenados con la inicial de su nombre de pila y su(s) apellidos(s), mientras que en B son almacenados con sus nombres completos. Podría ser una buena idea para este caso implementar una función hecha a medida $\Delta_?$ en la que dadas dos cadenas de texto $s = \{s_0, \ldots, s_k\}$ y $t = \{t_0, \ldots, t_k\}$ con el mismo número de tokens, y siendo cada token t_i definido como $t_i = t_{i,0}, \ldots, t_{i,n}$ donde $t_{i,j}$ es el caracter de posición j en el token t_i, si $|s_{0,1}| = 1$, $s_{0,1} = t_{0,1}$ y $s_i = t_i \forall i \in [1, k]$, entonces $\Delta_?(s, t) = 1$. Con esta definición, $\Delta_?$("A Perry", "Albert Perry") $= 1$.

Cualquier función que sea capaz de mapear dos cadenas de texto s y t a un número real $r \in [0, 1]$ debería encajar en el flujo de ejecución de MERA.

7.2.3. Bloqueo

MERA permite el uso de varias funciones de bloqueo al mismo tiempo. La definición de función de bloqueo, así como algunos de los ejemplos más representativos de este tipo de técnicas, han sido citados en la sección 6. No obstante, algunas de estas técnicas no son tan sencillas de incorporar al flujo de acción de MERA como las que hemos presentado en las secciones 7.2.1 and 7.2.2, puesto que pueden necesitar acceder a una estructura de datos extra al grafo G para alcanzar una ejecución eficiente.

Existen funciones de bloqueo cuyo funcionamiento no tendría que necesitar más estructuras de datos que el propio grafo, tal como la estrategia de considerar sólo como candidatos a hacer math aquellos elementos que coincidan exactamente en alguna propiedad semi-identificativa, como apellidos o fecha de nacimiento si tratamos con datos sobre personas [53]. Otras técnicas, como puede ser la indexación de q-gramas [4], requerirían la consulta de un índice de q-gramas. Esta nueva estructura no debería introducir ninguna inconsistencia con la información ofrecida por el grafo, sino que debería limitarse a presentar cierto rango de información útil para la función de bloqueo de una forma computacionalmente aceptable.

El algoritmo de MERA espera recibir un grafo G ya construido como entrada, que usará en modo de sólo lectura. Este hecho puede facilitar la construcción y mantenimiento de cualquier estructura extra T (como un índice de q-gramas), puesto que no debería T es accedida también en modo de sólo lectura ninguna de las estructura sería mutada durante la ejecución del linkado de entidades. Con asegurar una equivalencia de contenidos en el momento en que G ha sido construido por completo se aseguraría la consistencia entre G y T. Así, la estructura T podría construirse a la vez que se añaden datos a G o incluso explorando G una vez este ha sido ya completado y no se esperan más cambios hasta la ejecución del proceso de reconciliación.

No obstante, la construcción del grafo completo G puede ser un proceso no trivial, así como la creación de posibles estructuras extra. Sin embargo, las formas de afrontar dicho proceso se salen del ámbito de este trabajo y serán discutidas en trabajo futuro. La arquitectura presentada en esta memoria espera recibir dichas estructuras completas antes de la ejecución del linkado de fuentes.

Función de bloqueo propuesta. Hemos diseñado e implementado en un prototipo de MERA una función de bloqueo adaptada al manejo de formas

Algorithm 2 MERA Blocking

Require: Query q
Require: Index I
1: $N \leftarrow$ extractUniqueQrams(q.mainContent)
2: $E_{idf} = [\lambda]$
3: $E_{ngr} = [\lambda]$
4: **for all** $n_i \in N$ **do**
5: $\quad idf_{n_i} \leftarrow I$.idf($n_i$)
6: $\quad$ **for all** $e_i \in I$.entitiesWithQgram(n_i) **do**
7: $\quad\quad E_{idf}[e_i] \leftarrow E_{idf}[e_i]$+meraTfIdf($e_i, n_i, idf_{n_i}, I$)
8: $\quad\quad E_{ngr}[e_i] \leftarrow E_{ngr}[e_i] + 1$
9: result $\leftarrow \{\lambda\}$
10: **for all** $e_i \in E_ngr$ **do**
11: $\quad$ **if** $E_{ngr}[e_i] = |N|$ **then**
12: $\quad\quad$ result $\leftarrow$ result $+e_i$
13: **return** result $\cup$ bestK(E_{idf})

alternativas de una misma entidad, que consiste en una adaptación de indexación por q-gramas [4] y TF-IDF [60] que funciona de la forma descrita en el algoritmo 2. La eficacia de dicha función será discutida en la sección 9. No obstante, el pseudo-código aportado necesita ciertas clarificaciones para ser entendido por completo:

- Entrada *Query q*: consulta para la cual estamos buscando los candidatos más similares.
- Entrada *Index I*: estructura en la que se almacena información sobre q-gramas, sincronizada con el grafo G.
- Función *extractUniqueQgrams*: espera una cadena de texto s y devuelve un conjunto conteniendo todos los q-gramas $n_i \in s$.
- Propiedad *mainContent*: invocada sobre un objeto Query q, devuelve la cadena de texto principal de q.
- Estructura $\{\lambda\}$: representa un conjunto vacío.
- Estructura $[\lambda]$: representa un diccionario vacío en los que se almacenan pares de la forma (clave $\rightarrow$ valor).

- Método *idf*: invocado sobre un objeto Index I, espera recibir un q-grama n_i y devuelve la puntuación IDF de dicho q-grama en G (sincronizado con I).

- Método *entitiesWithNgram*: invocado sobre un objeto Index I, espera recibir un q-grama n_i y devuelve el identificador de aquello nodos en G asociados a alguna forma que contenga el q-grama n_i.

- Función *meraTfIdf*: espera un identificador de entidad e_i, un q-gram n_i, una puntuación IDF idf_{n_i} y un objeto Index I. Devuelve un resultado $r = tf \cdot idf_{n_i}$, donde tf es la cantidad de formas alternativas de e_i en las que el q-grama n_i aparece. Toda esta información es accesible a través de consultas a la estructura I.

- Función *bestK*: espera un diccionario de elementos y devuelve las claves de los k con mejor puntuación TF-IDF acumulada. El parámetro k es especificado por el usuario a través de configuración.

- *Iteración sobre diccionarios*: cuando accedemos a los elementos de un diccionario en *bucles for*, esperamos recibir en cada iteración la clave del par (clave $\rightarrow$ valor) correspondiente.

Las diferencias entre el algoritmo 2 descrito y las aproximaciones clásicas de TF-IDF basadas en q-gramas son las siguientes:

- Introducimos en el algoritmo el concepto de forma alternativa de una entidad. Una entidad es considerada como un conjunto de formas y el algoritmo tiene en cuenta cuántas de esas formas contienen cierto q-grama en lugar de cuántas veces aparece el q-grama en el total de las formas. Esto quiere decir que para la técnica descrita una entidad que sólo tuviese la forma asociada "'La, la, la", el bigrama "la" sería contado una única vez, porque aparece en una sola forma. En aproximaciones clásicas sería contado 3 veces por ser 3 el total de apariciones del bigrama.

- Manejamos la relevancia de los q-gramas objetivo en cada entidad mediante una puntuación de TF-IDF acumulado, pero también manejamos el número total de q-gramas encontrados en una misma entidad. Si cierta entidad contuviese todos los q-gramas de la cadena

q.mainContent, entonces dicha entidad promociona de forma automática a entidad candidata a la salida de la función de bloqueo, sin que sea tenido en cuenta su puntuación TF-IDF acumulada o incluso el límite de candidatos de salida fijado para la función en configuración.

Los resultados de esta técnica y los efectos de las novedades incluidas en ella respecto a aproximaciones clásicas serán discutidos en la sección 9.

7.3. Esquema y navegación de grafo

MERA está diseñado para resolver consultas contra un grafo en los que ciertos nodos representan entidades musicales. Devolver un resultado a una consulta en MERA requiere el uso de medidas de similitud entre cadenas de texto y navegación de grafo explorando el vecindario de los nodos entidad. MERA usa dos tipos de navegación de grafo para resolver una determinada consulta: búsqueda de formas alternativas dada una entidad y búsqueda de entidades relacionadas con dada.

7.3.1. Búsqueda de formas alternativas

Cada entidad (nodo) en un grafo de MERA puede ser asociada con varias formas identificativas o semi-identificativas. Consideremos un grafo G que contiene un nodo e_α que representa a la artista Lady Gaga. La identidad de la artista es representada por un único nodo e_α, pero e_α puede estar asociado con varios literales textuales que son los que realmente contienen las posibles formas identificativas de e_α. Tales cadenas de texto unidas mediante aristas (propiedades) con e_α podrían contener, entre otras contenidos, nombres canónicos o artísticos (“Lady Gaga”), nombre civil (“Stefani Joanne Angelina Germanotta”) o alias conocidos (“Mother Mosnter”, “Jo Calderone”). Puede que incluso erratas a la hora de escribir alguna de esas formas constituyan información útil a almacenar en G. Supongamos que existe cierta tendencia a escribir de forma incorrecta el nombre “Stefani”, usando en su lugar “Stefany” o “Stephanie”. Dependiendo de los algoritmos de reconciliación escogidos el hecho de que “Stephanie Joanne Angelina Germanotta” esté incluido en G puede resultar más o menos relevante (en este caso concreto, algoritmos fonéticos como Soundex encontrarían esta forma redundante), pero hay escenarios en que dicha información podría resultar útil. De hecho,

fuentes públicas de calidad sobre artistas almacenan y ofrecen como parte de sus datos ese tipo de variaciones [31].

Existen ejemplos de posibles formas identificativas de una entidad que requieren una navegación de grafo desde un determinado nodo más profunda que la exploración de propiedades a distancia de una arista. Un ejemplo se daría cuando manejamos artistas que además pertenecen a un grupo musical. Por cuestiones de metonimia, es posible encontrar canciones asociadas a un individuo cuando realmente deberían estar asociadas un grupo, o viceversa. Ejemplo de esto sería la vinculación entra la canción Imagine y la banda The Beatles, en lugar de con el artista John Lennon. Es por ello que podría tener sentido considerar como forma alternativa de un artista el nombre de los grupos a los que pertenece o ha pertenecido. O incluso todas las formas alternativas de dichos grupos.

El usuario de una implementación de MERA debe poder proporcionar por configuración qué formas alternativas de las que pueden encontrarse almacenadas en el grafo han de ser consideradas por cada tipo de entidad.

7.3.2. Búsqueda de entidades relacionadas

Una consulta en MERA está formada por un contenido principal, un tipo principal y un conjunto de posibles refinamientos. La salida de MERA al recibir una consulta q es un conjunto ordenado de nodos que representan entidades almacenadas en un grafo G que han sido detectadas como las más similares a lo especificado en q. Sin embargo dichos refinamientos podrían referirse a datos sobre otras entidades relacionadas de alguna forma con la entidad principal en q. Un ejemplo sería proporcionar como refinamiento de una consulta que trata de buscar una canción el título de su álbum.

Cada pieza de información extra incluida en q debe estar etiquetada, es decir, asociada a un tipo de datos que indique su naturaleza. Digamos que se está tratando de resolver una consulta q_α para encontrar un artista de nombre a, y que q_α además incluye como refinamientos dos nombres de canciones s_1 y s_2 asociadas al artista a. Como se ha descrito en el algoritmo 1 primeramente el sistema MERA seleccionaría un subconjunto de nodos S del grafo G contra el que estamos lanzando las consultas. Por cada nodo $n_i \in S$, MERA calcularía una pre-puntuación usando exclusivamente el contenido principal a de la consulta contra las diferentes formas alternativas del nodo. Tras ello, el sistema exploraría cada conjunto de entidades S'_i de tipo "canción" asociadas a cada nodo $n_i \in S$, y exploraría todas las formas alternativas

de cada entidad en S'_i para buscar coincidencias con s_1 y s_2. En caso de encontrar coincidencias en grado superior al definido por el usuario para el tipo "canción" cuando se ejecuta una consulta cuyo tipo principal es "artista", la puntuación del respectivo n_i sería incrementada. De esta forma, el cálculo de la puntuación de un determinado nodo contra una consulta implica explorar entidades relacionadas con dicho nodo en busca de coincidencias con los datos proporcionados en los refinamientos de la consulta.

No hay límite sobre el tipo y la cantidad de datos extra o refinamientos que pueden ser proporcionados en una consulta q. Puesto que trabajamos sobre grafos rdf, los tipos son especificados a través de propiedades rdf (o al menos a través de contenidos que una implementación de MERA pueda mapear a propiedades rdf de forma interna). Si se indicase un tipo t en q que no estuviese asociado a ninguno de los nodos candidatos o que incluso no apareciese en absoluto en el grafo, el sistema simplemente debería retornar un conjunto vacío de resultados.

7.3.3. Forma de grafo y manejo de fuentes

MERA está diseñado para permitir el manejo de diferentes diferentes fuentes de datos integradas en un mismo grafo, en el que se mantiene de qué fuente(s) procede cada pieza de información. Para ello se define un esquema de grafo rdf en el que los nodos no apuntan directamente a sus contenidos relacionadas, ya se trate de piezas de información literal u otras entidades con las que se vincula, sino que se relacionan con nodos auxiliares usados como intermediarios. Dichos nodos auxiliares unen el sujeto con el objeto a la vez que apuntan a un tercer nodo que identifica la fuente de datos de la cual ha sido extraída la información.

Supongamos que tenemos una canción s representada por un artista a. Una forma natural de representar dicha información en un grafo rdf sería la mostrada en la figura 7.2.

No obstante, también nos interesa saber *de acuerdo a quién* dicha canción s es representada por el artista a. Digamos que hemos podido hacer esa asociación gracias a los datos de cierto dataset d cuya información hemos volcado en nuestro grafo. La forma en la que MERA almacena este vínculo se muestra gráficamente en la figura 7.3. En términos rdf, una tripleta $t = (s, p, o)$ que ha sido obtenida de un dataset d es transformada en tres tripletas $t_0 = (s, p, n_{aux})$, $t_1 = (n_{aux}, p_{aux}, o)$ y $t_2 = (n_{aux}, p_{dat}, d)$, siendo n_{aux} un nuevo nodo auxiliar, p_{aux} el predicado especial "target" y p_{dat} el predicado "dataset".

Figura 7.2: Ejemplo de grafo sin esquema de MERA

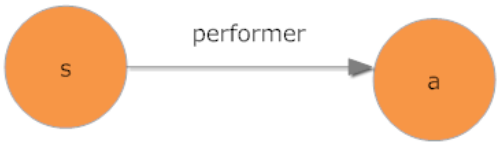


Figura 7.3: Ejemplo de grafo con el esquema de MERA

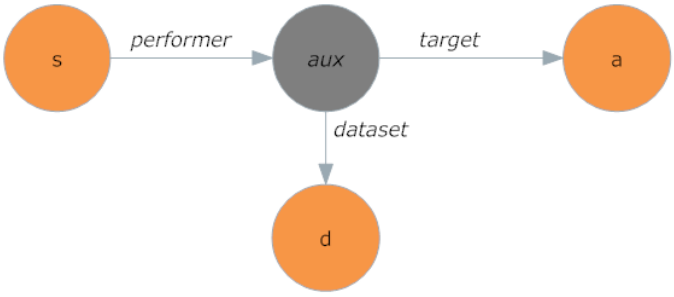
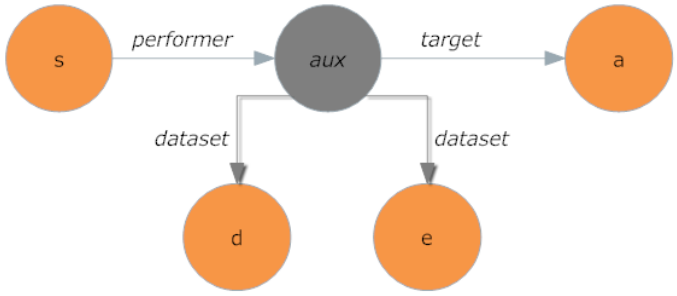


Figura 7.4: Nodo auxiliar asociado a varios datasets



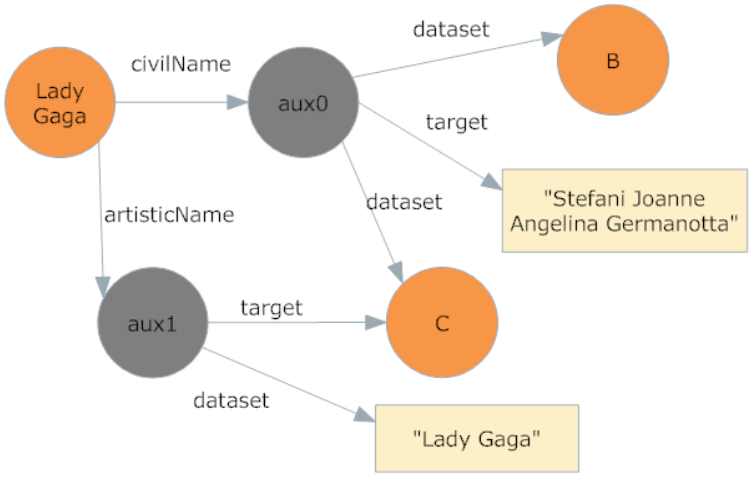
Como se muestra en la figura 7.3, s no apunta directamente a a, sino al nodo auxiliar aux, siendo aux quien está relacionado con a a través del predicado especial "target". A la vez, aux apunta al nodo d indicando de qué fuente de datos fue obtenida la tripleta original. En el esquema de grafo de MERA, los nodos nunca apuntan directamente a aquellos conceptos con los que estarían naturalmente unidos, si no que las relaciones se establecen a través de nodos auxiliares.

Supongamos que queremos añadir ahora la información de un nuevo dataset, al que nos referimos en nuestro grafo con un nodo e. De acuerdo al contenido de este nuevo dataset, la canción s también está representada por el artista a. El grafo resultante con el esquema propuesto por MERA sería el mostrado gráficamente en la figura 7.4. La relación que queremos representar es la misma que en el caso anterior, es decir, la tripleta básica que habría que añadir es (s,"performer", a). No obstante, ya existe un nodo auxiliar aux que representa dicho link. Por ello, solo añadimos al grafo la tripleta (aux,"dataset", e) para indicar que e también corrobora la relación representada con el nodo aux entre a y s.

Suponemos dos usos principales a la información de qué fuentes proveen cada pieza de dato presenten en el grafo. Por un lado, MERA permite al usuario a través de configuración el uso de listas blancas/negras seleccionar qué fuentes han de ser usadas durante el proceso de matching, de forma que sólo ciertas regiones del grafo sean consideradas. Por otro lado, el usuario podría filtrar los resultados de MERA una vez recibidos de acuerdo a su criterio.

Un ejemplo motivacional de por qué filtrar datos de otras fuentes cuando han sido recibidos los resultados en lugar de excluir las fuentes durante el proceso de matching, permitiendo de esa forma reducir el número de com-

Figura 7.5: Artista descrito en dos fuentes de datos



paraciones necesarias por operar con regiones reducidas del grafo, sería el siguiente: supongamos que descubrir links entre dos fuentes de datos A y B, donde la información de A es usada para construir consultas y la información de B es volcada a un grafo G con el esquema propuesto por MERA. El conjunto A contiene una entidad artista identificada con la cadena "Lady Gaga", mientras que G, construido a partir de los datos de B, contiene un nodo b_α que representa a la artista Lady Gaga, pero cuya única forma asociada es "Stefani Joanne Angelina Germanotta". Antes de tratar de establecer link, enriquecemos el grafo G usando una tercera fuente de datos C que incluye información sobre artistas, almacenando para una misma persona sus nombres artístico y civil. Al enfrentar C contra G, la entrada en C que representa a la artista Lady Gaga se detecta como un match con el nodo b_α, lo que nos permite volcar a G nueva información sobre ese nodo aportada por C. Así, llegamos a construir un grafo G' como el que se muestra en la figura 7.5

Nuestro objetivo es detectar links entre las fuentes A y B. No obstante, si configurásemos MERA para funcionar con listas blancas teniendo sólo en cuenta la información presente en el grafo de B, nos encontraríamos en una situación en la que no habría información suficiente para descubrir un vínculo entre la entrada "Lady Gaga" en la fuente A y la entrada la única forma "Stefani Joanne Angelina Germanotta" asociada al nodo Lady Gaga en G si sólo usamos las tripletas de B. Por otro lado, usando todas las tripletas disponibles, el match con el nodo de Lady Gaga sería posible. Al recibir los

resultados, el usuario puede comprobar si la información recibida contiene datos de la fuente B (aunque no hayan sido los datos usados para encontrar el link) o si por contra todos los daros proceden de otras fuentes. En caso de que haya parte de información de B, como en el escenario que estamos describiendo, el resultado sería pertinente. En caso contrario, el resultado se ignoraría, puesto que se desviaría del objetivo inicial de encontrar links entre entradas de A y B.

7.3.4. Creación de grafos

El proceso de construir un grafo que represente la información de una fuente cuyos datos no son originalmente presentados con un esquema de grafo o similar no es una tarea trivial. Sin embargo, el grado de dificultad de este proceso puede variar entre los diferentes casos particulares. La unión de varias fuentes de datos en un único grafo sin que estas compartan identificadores únicos representa un escenario problemático debido a que sería necesario aplicar técnicas de detección de equivalencias entre ambas fuentes como la que presentamos en este trabajo. Como escenario opuesto, la creación de un grafo a partir de los datos de una única fuente que cuente con identificadores internos únicos para definir la identidad de sus entradas puede resultar más sencillo, pues no debería existir ambigüedad en la identidad de cada elementos.

En el presente trabajo asumimos que el grafo con el que se trabaja se recibe como parámetro, sin discutir las mejores formas de construir dicho grafo. No obstante, pretendemos discutir en trabajos futuros sobre las formas más adecuadas de generar este tipo de estructuras.

7.4. Configuración

En la tabla 7.1 se han recogido y explicado la mayoría de opciones de configuración planteadas por MERA.

7.4.1. Configuración de comparaciones

Como se muestra en la tabla 7.1, la mayoría de opciones de configuración de MERA están referidas a diferentes estrategias y valores a usar en los diferentes escenarios de reconciliación, permitiendo al usuario especificar

Cuadro 7.1: Tabla de configuraciones

Concepto	**Rango**	**Defecto**
Valores por defecto generales		
• **Por tipo de entidad**		
— (D1) Algoritmos estandarización	Lista de algors.	null
— (D2) Algoritmos comparación	Lista de algors.	Match exacto
— (D3) Umbral	[0,1]	0.65
— (D4) Relevancia	[0,1]	0.70
• **Por tipo de orden**		
— (D5) Umbral tipo principal	[0,1]	0.65
— (D6) Umbral total	$\mathbb{R}^+$	0.95
Valores por defecto en tipos		
— (D7) Algoritmos estandarización	Lista de algors.	D1
— (D8) Algoritmos comparación	Lista de algors	D2
— (D9) Umbral	[0,1]	D3
— (D10) Relevancia	[0,1]	D4
Bloqueo		
— Máximos candidatos por bloqueo	$\mathbb{N}^+$	todos
— Máximos resultados por consulta	$\mathbb{N}^+$	10
— (D11) Funciones de bloqueo	Lista de funciones	null
Filtrado de fuentes		
— Tipo	{Lista blanca, Lista negra, null}	null
— Lista de fuentes	Conjunto de URIs	null
Configuración de órdenes		
— Umbral tipo principal	[0,1]	D5
— Umbral total	$\mathbb{R}^+$	D6
— Algoritmos estandarización	Lista de algors.	D1
— Algoritmos de comparación	Lista de algors.	D2
— Funciones de bloqueo	Lista de funciones	D11
• **Por tipo de refinamiento**		
— Algoritmos estandarización	Lista de algors.	D7
— Algoritmos comparación	Lista de algors.	D8
— Umbral	[0,1]	D9
— Relevancia	[0,1]	D10

conjuntos de algoritmos, umbrales de similitud y relevancia de tipos. El conjunto de valores especificado en la sección de la tabla *Valores por defecto generales* es usado en caso de que el usuario no los sobrescriba para casos de órdenes o tipos específicos. El significado de cada una de esas opciones se explica en la siguiente lista:

- *Por tipo de entidad*. Estamos definiendo valores para usar cuando un dato de tipo t se utiliza para refinar una query con un tipo principal $o \neq t$.
  - *Algoritmos estandarización*. Conjunto de algoritmos a usar en la etapa de pre-procesamiento.
  - *Algoritmos comparación*. Conjunto de algoritmos a usar en la etapa de comparación.
  - *Umbral*. Cuando comparamos dos entidades de tipo t como refinamiento de una consulta, si ninguno de los algoritmos de comparación usados devuelve un valor mayor o igual que esta cifra la puntuación obtenida será descartada, es decir, no se sumará en ningún caso a la puntuación principal de la consulta.
  - *Relevancia*. Cuando comparamos dos entidades de tipo t como refinamiento de una consulta, en caso de que se encuentre un match lo suficientemente alto (puntuación $\geq$ umbral), la puntuación del refinamiento será sumada a la puntuación general de la entidad principal tras ser ponderada multiplicando por el factor de relevancia configurado.
- *Por tipo de orden*. Estas opciones se aplican al score general de una query contra una entidad o a las operaciones con el tipo principal de la consulta.
  - *Umbral tipo principal*. Los matches con entidades cuya puntuación esté por debajo de este umbral serán descartados sin entrar a valorar refinamientos.
  - *Umbral total*. Si el resultado de comparar el tipo principal de una consulta más la puntuación de todos los refinamientos obtenidos no supera este valor, el match se descarta.

- No se proporcionan algoritmos por defecto para operar con el tipo principal de las órdenes. Los algoritmos por defecto del tipo principal de una orden serán aquellos que el tipo tenga asignados como por defecto en caso de comparar cadenas en refinamientos.

Una implementación de MERA debe seguir una configuración de valores por defecto jerárquica con un esquema de herencia como el mostrado en la tabla 7.1. No obstante, el usuario debe poder sobrescribir los valores de heredados en diferentes niveles de profundidad, permitiendo incluso que comparaciones de cadenas del mismo tipo se ponderen diferente dependiendo del tipo principal de la consulta que haya desencadenado dicha comparación. Ejemplo: un usuario podría especificar que cuando se ejecuta una consulta para buscar artistas, si se facilitan nombres de álbumes como refinamientos estos tengan un peso de 0.50 y un umbral de 0.80. Sin embargo, el usuario debería también de poder configurar que en caso de estar ejecutando una consulta para encontrar canciones, si se facilita el nombre de un álbum como refinamiento el peso sea de 1.0 y el umbral sea de 0.70.

7.4.2. Reduciendo el número de resultados

Los usuarios de una implementación de MERA han de poder especificar el número de entidades máximas a devolver en los resultados de una query o al finalizar la fase de bloqueo de candidatos. Los candidatos por defecto a la salida de la etapa de bloqueo al ejecutar una consulta serían *aquellos elementos en G de tipo o principal de la consulta*, puesto que la estrategia de bloqueo por defecto es no usar estrategia de bloqueo. No obstante, este parámetro debe ser modificado si se incluye una función de bloqueo.

7.4.3. Filtrado de fuentes

El comportamiento por defecto de MERA es el de usar todos los datos disponibles en el grafo, sin aplicar ningún filtro de fuentes, pero una implementación de MERA debe permitir al usuario establecer listas blancas/negras de fuentes de datos a usar durante el proceso de reconciliación, seleccionando así regiones del grafo a utilizar durante el proceso de reconciliación. La forma de identificar esas fuentes sería a través de la URI que las identifica en el grafo, o la de proporcionar contenido que una implementación de MERA pueda mapear a tales URIs.

7.4.4. Especificación y adición de algoritmos

El diseño de MERA está pensado para permitir al usuario desarrollar e incluir en el flujo de ejecución diferentes algoritmos en cada una de las etapas de reconciliación, incluyendo funciones codificadas por el usuario además de un posible rango de opciones ofrecido por una implementación de MERA. Del mismo modo, una implementación de MERA debe permitir al usuario referirse a esos nuevos algoritmos en configuración.

Por un lado, una implementación de MERA en la que la configuración se espera a través de archivos podría usar mecanismos como las convenciones de nombres para referirse al código incorporado por el usuario. Por contra, una implementación en la que la configuración se espera a través de código podría solventar esa situación mediante el uso de callbacks. El prototipo que hemos implementado y presentado en la sección 10 se configura a través de archivos JSON, siendo estos algoritmos especificados a través del nombre de su clase. Dicho nombre de clase es mapeado a la ruta de un archivo donde se encuentra la implementación de una clase que cumple el pertinente interfaz de algoritmo.

7.4.5. Definición de formas alternativas

Una opción configurable no mostrada en la tabla 7.1 es la especificación de formas alternativas por cada tipo de nodo. Resulta difícil ofrecer una configuración por defecto de este tipo de valores, pues las formas alternativas de entidad están estrechamente ligadas tanto a la naturaleza de la entrada concreta a la que nos referimos como a las ontologías usadas para representar la información en el grafo rdf. Por ejemplo, posibles formas alternativas del tipo “artista” serían “nombre civil”, “nombre artístico”, “alias”, “variación de nombre”... el usuario debe poder especificar como formas alternativas piezas de información accedidas a través de navegaciones de grafo complejas. Para el caso de “artista”, un ejemplo sería “variaciones de nombres de grupos a los que pertenezca ”, lo que implicaría navegar desde un nodo “artista” a aquellos nodos “grupo” con los que esté relacionado y, desde los nodos grupo, acceder a cadenas de texto linkadas con predicados “variación de nombre”.

7.4.6. Valores especificados por defecto

Los valores numéricos mostrados en la tabla 7.1 son simplemente un conjunto funcional de valores que libera al usuario de configurar todas las opciones de MERA. Sin embargo, desde el punto de vista de que la motivación principal de este trabajo es la dificultad para encontrar escenarios de reconciliación estándar de forma que cierta combinación de algoritmos, umbrales y relevancias encaje en todas las situaciones, los resultados podrían mejorar potencialmente con configuraciones adaptadas a cada caso particular.

Capítulo 8

Experimento

Hemos implementado un prototipo usando el lenguaje de programación python que recoge la mayor parte de las especificaciones de MERA, incluyendo:

- Exploración de grafo para buscar formas alternativas de una misma entidad.
- Exploración de grafo para buscar entidades relacionadas a una dada.
- Configuración de factores de relevancia al aplicar refinamientos a una consulta.
- Configuración de umbrales de aceptación para los distintos tipos, tanto para el contenido principal de una consulta como para sus distintos refinamientos.
- Posibilidad de incluir la función de bloqueo descrita en la sección 7.2.3.
- Uso del esquema de grafo de MERA propuesto en la sección 7.3.
- Conjunto de algoritmos de comparación de propósito general.
- Conjunto de funciones de estandarización de texto.

Hemos usado nuestro prototipo para determinar la validez de las propuestas de MERA. Nuestro experimento ha consistido en la reconciliación de una parte aleatoria de dos fuentes de datos diferentes A y B contra una tercera fuente C. Hemos puesto la información de C en un grafo G y hemos usado

las entradas de A y B para la elaboración de consultas. Hemos comprobado de forma manual cual debería ser el resultado esperado para cada una de las consultas construidas con los datos de A y B y hemos medido la corrección de los resultados arrojados por nuestro prototipo para cada consulta bajo diferentes condiciones.

La fuente de datos A ha sido escogida por contener potencialmente datos de calidad elevada, es decir, con pocas o ninguna errata y con asociaciones fiables y numerosas entre entidades. Por otro lado, la fuente de datos B ha sido escogida por contener datos potencialmente ruidosos, es decir, con frecuentes erratas, palabras vacías y con relaciones entre entidades incompletas. No obstante, todas y cada una de las consultas seleccionadas al azar incluyen al menos una entidad principal y un refinamiento para la misma, aunque no se ha hecho uso de dicho refinamiento en todos los experimentos. Todas las consultas tratarán de encontrar nodos de tipo "'canción" en el grafo, y todos los refinamientos se referirán a artistas o escritores relacionados con dicha canción.

Hemos lanzado las mismas consultas varias veces contra G usando nuestro prototipo, excluyendo/incluyendo en las distintas iteraciones algunas de las características de MERA para comprobar el efecto que esto tiene sobre el resultado. Hemos mantenido la misma selección de algoritmos, umbrales y relevancias durante todos los experimentos, puesto que pretendemos comprobar el efecto que la inclusión/exclusión de características de MERA tiene sobre el resultado, no la efectividad de los algoritmos concretos usados.

Cuando nuestro prototipo recibe una consulta q, devuelve un ranking de nodos en G que han sido detectados como más parecidos a G siempre y cuando cumplan las mínimas condiciones configuradas sobre umbrales de similitud. Nuestra medición de resultados ha consistido en detectar en que posición ha quedado la entidad que se identificó de forma manual que debería ser el resultado de la consulta, si es que ha sido detectada.

8.1. Descripción de los experimentos

8.1.1. Condiciones de los experimentos

Hemos ejecutado nuestro experimentos bajo las siguientes condiciones de software/hardware:

- Máquina Virtual usando los servicios de computación en la nube de

Windows Azure.

- Sistema Operativo de 64-bit: Windows Server 2012 R2.
- Procesador AMD Opteron (tm), 4171 HE 2.10 GHz.
- 14 GB de memoria RAM.
- Python 2.7.3 64-bit.

8.1.2. Selección de datos

Grafo

Hemos usado la fuente de datos Discogs [31] para la creación de un grafo G conteniendo un total de 500.000 canciones, así como sus artistas y escritores asociados. La información para construir G ha sido extraída del volcado de datos de Discogs publicado el 1 de enero de 2015. Se han aprovechado los archivos de lanzamientos (releases) y artistas (artists) disponibles en dicho volcado.

El archivo de lanzamientos se estructura de forma que cada release es asociada con las pistas (tracks) que contiene. Hemos detectado un total de 45.458.287 pistas entre todas las releases publicadas. La escalabilidad de nuestro prototipo no es suficiente para manejar un grafo que contenga los datos completos de los más de 45 millones de pistas, puesto que el grafo es por completo manipulado en memoria principal. Por ello, hemos reducido esa cantidad a 500.000, una cifra que consideramos representativa y que podemos manejar. Las canciones seleccionadas incluyen aquellas pistas que han sido manualmente detectadas como el resultado esperado a las consultas. El resto de elementos hasta llegar a 500.000 ha sido seleccionado de forma aleatoria mediante el uso de la función de python *random.randInt()* para generar números al azar entre 1 y 45.458.287 (cantidad total de pistas detectadas). Las canciones con una posición relativa dentro del volcado de datos que estén contenidas en el conjunto de números aleatorios generado fueron incluidas en los datos del grafo.

También hemos incluido en el grafo G los datos completos de los diferentes artistas y escritores asociados a los 500.000 trabajos. Discogs asocia personas o grupos a pistas/lanzamientos a través de diferentes roles, Hemos mapeado algunos de los roles de Discogs a nuestros propios roles para incluir dicha

información en G. Los roles de Discogs "Artist", "Featuring", y "Vocals" han sido mapeados a "artista", y el rol "Written-By" ha sido mapeado a "escritor". El resto de personas asociadas a las canciones con diferentes roles de Discogs ha sido ignorado.

Todos los lanzamientos y artistas tienen su identificador único interno en la fuente de datos Discogs, y dentro de cada lanzamiento las pistas tienen una posición única asociada. Por ello, no ha sido necesario aplicar mecanismos de reconciliación entre entradas a la hora de volcar los datos de Discogs a G. En total, el grafo usado contiene un total de 500.000 canciones y 624.441 artistas relacionados.

Fuente de alta calidad

La enciclopedia musical on-line MusicBrainz encaja con nuestro requisitos de ofrecer una gran cantidad de metadatos musicales de alta calidad [67]. Se han ido extrayendo trabajos de MusicBrainz al azar de entre las categorías de "work" y "record", comprobando de forma manual que las entradas obtenidas también se hallaban entre los datos del volcado de Discogs. Hemos repetido este proceso hasta alcanzar la cantidad de 100 canciones presentes en ambas fuentes y hemos usado estas entradas en Discogs para la generación de consultas en nuestro experimentos. La media de artistas/escritores usados como refinamiento en cada consulta ha sido de 2.62. Los datos seleccionados aleatoriamente para la generación de consultas se encuentran en el anexo 13.1.

Fuente ruidosa o de baja calidad

En 2006, , AOL Inc. libera un archivo conteniendo unas 20 millones de entradas introducidas por unos 650.000 usuarios en su motor de búsqueda [3] a lo largo de un periodo de tiempo de 3 meses. Hemos considerado que las búsquedas relacionadas con entidades musicales que pudiésemos encontrar dentro de este contenido podrían ser un ejemplo representativo de fuente de datos ruidosa, pues todas las imprecisiones, erratas o incorrecciones encontradas no habrían sido generadas de forma artificial, sino que han sido introducidas por usuarios reales anónimos sin estar condicionados por el experimento.

Nuestra estrategia para encontrar búsquedas relacionadas con contenido musical, que además contengan al menos una canción y un artista asociado,

ha sido la siguiente:

- Hemos buscado entre todas las búsquedas de usuario aquellas que contuviesen las palabras "feat" o "featuring". Hemos separado cada uno de los items encontrados en dos partes, usando "feat" como separador y excluyendo esta palabra de cada uno de los trozos obtenidos, para añadir cada parte a un conjunto de claves candidatas ruidosas N.

- Hemos hecho una limpieza automática de todas las claves en N borrando aquellas palabras o secuencias carentes de significado para nuestros propósitos, como "to download", "ringtone", "free music", "song of", "video", etc. En caso de encontrar el caracter " - " en la clave, hemos partido el elemento usando "-" como separador y repetido el proceso de limpieza en cada una de las partes resultantes. Todos los resultados obtenidos que no contuviesen al menos algún caracter alfanumérico fueron añadidos a un conjunto C de claves candidatas limpias. El tamaño alcanzado por C fue de 1203 elementos.

- Hemos revisado de forma manual cada clave en C en el contexto de su aparición en el log de AOL para comprobar si realmente se estaban refiriendo a artistas/canciones o no. Además, si detectamos que algunas de las canciones contenía más de un artista/canción a la vez, la hemos separado en partes y considerado cada entidad de forma independiente. Al final de este proceso, 922 secuencias (no únicas) fueron detectadas como artistas/canciones. Almacenamos todas ellas en un conjunto de claves encontradas F.

- Incluimos todos los elementos de F en un conjunto de claves definitivas D. Tras ello, para cada clave $k_i \in F$ con una longitud mayor que 3, generamos todas las posibles cadenas de texto $S_i = \{s_0, s_1 \dots s_k\}$ en una distancia de Levenshtein [46] de 1 con k_i. Tras ello, añadimos a D cada elemento $s_j \in S_i$, siempre y cuando s_j fuese encontrado más de 5 veces en el log de AOL. Con esto, hemos obtenido un grupo de claves que representan entidades musicales que fueron introducidas por usuarios anónimos, así como variaciones de caracteres de las mismas que detectamos que aparecían entre las consultas con una mínima frecuencia, aunque desconociésemos si aparecían como parte de una búsqueda de entidades musicales o no. Borramos del conjunto D secuencias no

significativas para nuestros propósitos como las descritas en pasos anteriores.

- Exploramos de nuevo el log completo de AOL seleccionando aquellas búsquedas que contuviesen alguna clave de D. Elaboramos tres lista diferentes con las entradas encontradas, repartiendo los elementos según contuviesen una sola clave de la lista, entre dos y tres, o más de cuatro. 327.904 búsquedas fueron detectadas.

- Desordenamos las listas obtenidas con la función de python *random.shuffle()*. Tras ello, revisamos manualmente dentro las listas para obtener los primeros 35 resultados de cada una que se pudieron identificar como búsquedas de canciones que incluyesen, al menos, un nombre de artista asociado. Borramos de los resultados todas las palabras vacías de significado para nuestro propósito de igual forma que en pasos anteriores.

En todos los casos, hemos dejado sin modificar la aparición original de cada entidad en el log de AOL. Nuestro tratamiento ha consistido en la detección de entidades, separando las búsquedas en partes si procedía, y en el borrado de palabras identificadas como vacías de significado, pero ningún otro cambio fue aplicado sobre el contenido original.

La separación en diferentes listas de consultas que contuviesen una de las claves de D, entre dos y tres o más de cuatro fue realizada con objeto de asegurar cierta presencia de consultas que contuviesen más de un artista asociado a la vez. La media final de artistas asociadas a cada canción en las consultas elaboradas fue de 1.44. Los datos seleccionados aleatoriamente para la generación de consultas se encuentran en el anexo 13.2.

8.1.3. Diseño de experimentos

Hemos ejecutado una serie de experimentos diseñados para medir el efecto que tienen las propuestas de MERA sobre escenarios con diferente calidad de datos. En las diferentes iteraciones de los experimento hemos incluido diferentes subconjuntos de características de MERA para comprobar qué grado de impacto tiene cada una de ellas sobre el resultado final. Para todos los experimentos realizados hemos utilizado la misma configuración, si fuese procedente, es decir, si la característica de MERA configuraba se encontraba activa durante el experimento.

- Función de bloqueo basada en indexación de q-gramas y TF-IDF. Se calculará un índice de *tri-gramas* para todas las entidades presentes en el grafo, donde se indicará el número de veces que aparece cada tri-grama en el corpus, qué entidades contienen cada tri-grama y cuál es el número total de entidades de cada tipo

- Función de bloqueo configurada para seleccionar un máximo de 60 candidatos. Sólo en caso de estar activa la función de bloqueo propuesta en este trabajo, tal y como está descrita en la sección 7.2.3 esa cifra podrá ser excedida.

- Número máximo de resultados por consulta: 15.

- Umbral mínimo de canción para ser tenida en cuenta en los resultados: 0.50.

- Umbral mínimo de un artista para ser tenido en cuenta en los resultados: 0.65.

- Relevancia de un refinamiento por artista: 0.80.

- Algoritmos de reconciliación usados en todos los casos: similitud de Levenshtein.

- Funciones de estandarización incluidas: sustitución de todos los caracteres no ASCII por equivalentes en ASCII con la función de python unicodedata.normalize(“NFKD”, targetString) , paso a minúsculas de todas las entradas, sustitución por espacios en blanco de todos los signos de puntuación and borrado de espacios en blanco redundantes (a principio o final de cadena o varios consecutivos).

Esta configuración ha sido pensada para permitir la aparición de resultados relativamente malos (en un ranking bajo) entre los resultados debido a los umbrales bajos de artista y canción y al amplio número de candidatos en los resultados, puesto que nuestro interés exclusivo es el de descubrir en qué posición termina el nodo detectado manualmente como match objetivo de cada consulta. Una ejecución de MERA que busque matches seguros en un entorno real probablemente deba definir umbrales de aceptación de candidatos más altos.

La razón por la que hemos escogido un único algoritmo de comparación de cadenas es que queremos comprobar cómo las propuestas de MERA, más allá de la efectividad de los algoritmos concretos seleccionados en cada caso. Las formas alternativas que hemos considerado para cada entidad a la hora de construir el grafo G que contiene los datos de la fuente Discogs han sido los siguientes:

- Artistas/escritores: nombre canónico de acuerdo a Discogs, nombre civil, aliases y variaciones de nombre habituales. En caso de trabajar con un grupo, también el nombre canónico de todos sus integrantes. En caso de tratar con un persona, el nombre canónico de los grupos a los que pertenece.

- Canciones: nombre canónico de acuerdo a Discogs. En caso de que dicho nombre contenga texto entre paréntesis (ejemplo: "Summer love (radio remix)"), el mismo texto habiendo sido borrada la información entre paréntesis será considerada una forma alternativa.

Las siguientes combinaciones de capacidades de MERA han sido probadas:

- *Experimento A*: reconciliación de canciones sin hacer uso de navegación de grafo. Función de bloqueo basada en indexación de tri-gramas y puntuación TF-IDF. Los resultados se muestran en el gráfico 8.1a.

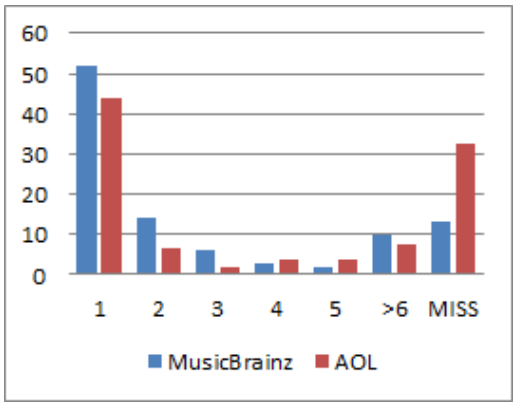
- *Experimento B*: Reconciliación buscando formas alternativas de canciones, pero sin usar información de artistas relacionados. Función de bloqueo basada en indexación de tri-gramas y puntuación TF-IDF. Los resultados se muestran en el gráfico 8.1b.

- *Experimento C*: Reconciliación buscando en el grafo refinamientos por artistas relacionados, pero sin usar formas alternativas para ninguna entidad. Función de bloqueo basada en indexación de tri-gramas y puntuación TF-IDF. Los resultados se muestran en la gráfica 8.1c.

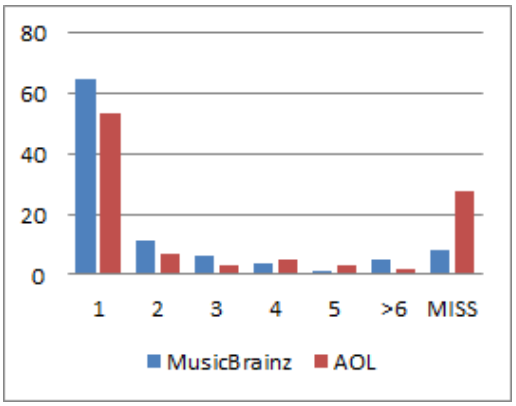
- *Experimento D*: Reconciliación usando ambos tipos de navegación de grafo (formas alternativas y entidades relacionadas). Función de bloqueo basada en indexación de tri-gramas y puntuación TF-IDF. Los resultados se muestran en el gráfico 8.1d.

- *Experimento E*: Reconciliación usando ambos tipos de navegación de grafo y la función de bloqueo descrita en la sección 7.2.3. Los resultados se muestran en la gráfica 8.1e.

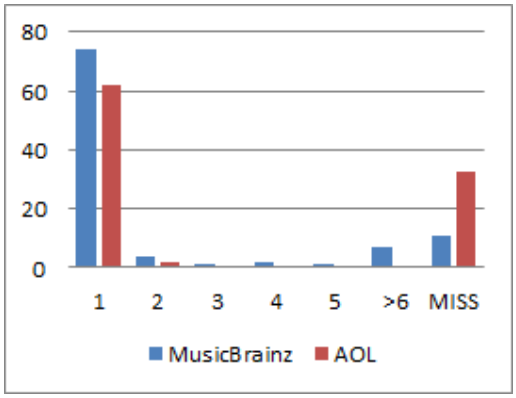
Además, hemos incluido una comparación de la diferencia existente entre usar una estrategia clásica de TF-IDF + indexación de tri-gramas con ambas estrategias de navegación de grafo activas y usar la función de bloque descrita en la sección 7.2.3. Los resultados para AOL son mostrados en el gráfico 8.2a y los resultados para MusicBrainz en el gráfico 8.2b.



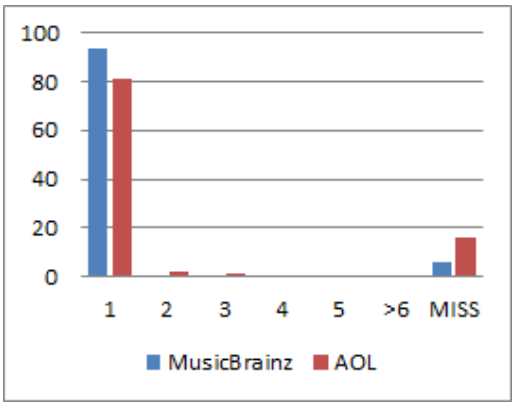
(a) Sin navegación de grafo



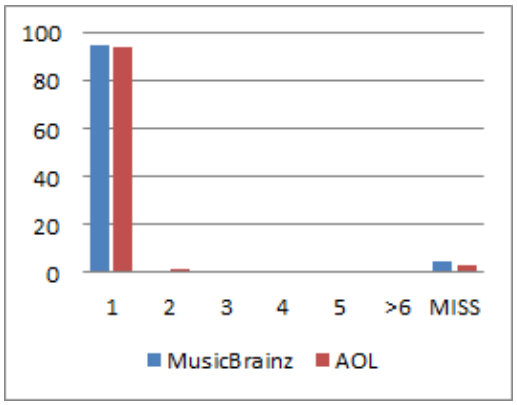
(b) Navegación por formas alternativas



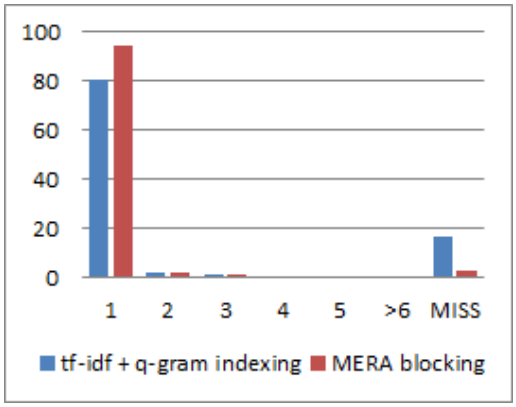
(c) Navegación por entidades relacionadas



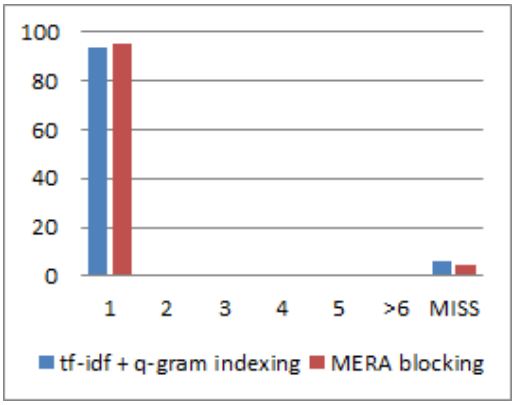
(d) Ambos tipos de navegación



(e) Both navigation ways y bloqueo MERA



(a) Estrategia de bloqueo en AOL



(b) Estrategia de bloqueo en MusicBrainz

Capítulo 9

Discusión

9.1. Inclusión de características

9.1.1. Experimento A. Sin navegación de grafo

Como se muestra en el gráfico 8.1a, el 52 % de las consultas de MB[1] han sido detectadas en primer lugar, el 35 % han sido detectadas en posiciones intermedias y el 13 % no han sido detectadas. En el caso de AOL, el 43.81 % en primera posición, el 23.81 % en posiciones intermedias y el 32.38 % no detectados.

Hemos analizado el por qué de los casos particulares para los que el prototipo arrojó o bien resultados intermedios o bien ni siquiera detectó el nodo objetivo, y nos hemos encontrado con las mismas razones en general para ambas fuentes:

- El nodo objetivo no pasó el corte de la función de bloqueo. Esto ha ocurrido principalmente al tratarse de canciones de nombre corto, con tri-gramas comunes en el corpus o por combinación de estos dos factores. Un ejemplo de esto sería la canción "So what" de la artista "Ciara" o la canción "Bestfriend" del artista "50cent", ambas consultas de AOL.

- Ambigüedad: En algunos casos, los nombres de canciones no son únicos. Sin explorar artistas relacionados para decidir qué resultado es mejor, no existe un criterio confiable para decidir cuál de todas las canciones con nombre idéntico debería aparecer en primer lugar. Un ejemplo de

[1] MusicBrainz

esto sería la canción “SOS” de la artista “Rihanna”, pues muchas otras canciones con nombre “SOS” han resultado ser incluidas en el grafo.

- Falta de formas alternativas de canción. Los títulos de canciones en AOL y MB presentan en ocasiones una gran diferencia con su equivalente en Discogs de acuerdo al criterio de la similitud de Levenshtein, a pesar del hecho de que ambas entradas identifican la misma realidad. Esto es debido a que ocurre con cierta frecuencia que los nombres de pistas en Discogs añaden contenido entre paréntesis al nombre principal de la canción, como “Summer love (radio edition)” o “Summer love (remix DJ)”. Esta información puede ser útil para propósitos de desambiguación y de hecho lo es en algunos casos, como cuando se intenta distinguir la versión original de una canción con un remix concreto, pero también puede actuar como *ruido* cuando tales cadenas se intentan reconciliar con fuentes no tan precisas como MB o las consultas extraídas del log de AOL.

9.1.2. Experimento B.Navegación de grafo en búsqueda de formas alternativas

Como se muestra en el gráfico 8.1b, existe una mejora en los resultados para ambas fuentes comparados con los datos obtenidos en el experimento A. La tasa de resultados detectados en primer lugar para AOL crece de 52 % a 65 %, la de resultados intermedios desciende de 14 % a 11 % y la de resultados perdidos desciende de 13 % a 8 %. En el caso de AOL, la tasa de primeros resultados crece de 43.81 % a 53.33 %, la de intermedios disminuye de 23.81 % a 19.05 % y la de perdidos disminuye de 32.38 % a 27.62 %.

Analizando cada caso particular, nos encontramos con que, de nuevo, las razones de mejoría para ambas fuentes han sido prácticamente las mismas. El uso de la exploración del grafo para detectar formas alternativas de canciones nos ha permitido relacionar entradas como “Touch it” de AOL con equivalente en Discogs como “Touch it (remix radio edition)”, ya que el texto entre paréntesis ha sido borrado para introducir una forma alternativa de la canción en el grafo. Además, la función de bloqueo varía la lista de candidatos devuelta respecto al experimento A, puesto que el uso de formas alternativas hace que se modifiquen los tri-gramas que contienen aquellas entidades que han sido completadas así como los pesos de los tri-gramas en el corpus. No obstante, la inclusión de esta característica también ha tenido como efecto la

aparición de falsos positivos que no se habían dado en el experimento A. A pesar de que existe una mejoría de resultados en términos generales, algunas de las entradas concretas empeoran su ranking debido a que entidades que no habían sido detectadas como matches (y que no deberían haberse detectado) en el experimento A, mediante la inclusión de su forma alternativa han mejorado en ranking.

9.1.3. Experimento C. Navegación de grafo en búsqueda de entidades relacionadas

Como se muestra en el gráfico 8.1c, cuando usamos exploración de grafo para buscar artistas relacionados con la canción de la consulta, los resultados mejoran comparados con el experimento A. En el caso de MB, la tasa de primeros resultados crece de 52 % a 74 % y la de intermedios baja de 35 % a 15 %. En el caso de AOL, la tasa de primeros resultados crece de 43.81 % a 61.9 % y la de intermedios desciende de 23.81 % a 5.7 %.

Para ambas fuentes, los resultados del experimento C han resultado ser mejores que los del experimento B. La conclusión que sacamos de este hecho es que la navegación de grafo en búsqueda de artistas relacionadas tiene en general más impacto que la navegación en busca de formas alternativas. En el caso de AOL, la tasa de primeros resultados crece hasta 53.33 % en B frente al 61.9 % obtenido en C. Para MB, la tasa de primeros resultados crece hasta 65 % en B frente al 74 % de C. No obstante, no existen una mejora significativa, o incluso ninguna mejora en absoluto en el caso de AOL, respecto a la tasa de entidades no encontradas si comparamos los experimentos A y C. Es decir, la inclusión de navegación de grafo buscando artista relacionados prácticamente no ha influido en la detección de artistas que no fueron siquiera detectados por el prototipo sin navegación de grafo. En cambio, gran parte de aquellas entidades que en el experimento A clasificaron en posiciones intermedias mejoran su ranking en C, ascendiendo en notables ocasiones a la primera posición.

La razón encontrada para la mejora de la tasa de primeros resultados para ambas fuentes es que en los casos de nombres idénticos o muy similares de canción en los que carecíamos de criterio para decidir qué entrada debería clasificar mejor, ahora tenemos nombres de artistas como mecanismo desambiguatorio. Algunos de los resultados no son aún encontrados debido a la carencia de formas alternativas, sobre todo de canciones con información

entre paréntesis, puesto que la longitud de esas secuencias extra marcan la diferencia de acuerdo al criterio de la similitud de Levenshtein. El hecho de que la tasa de entidades perdidas no varíe prácticamente comparada a la del experimento A se debe a que aquellas entidades que no pasaron el corte de la etapa de bloqueo en A siguen en pasarlo en C, puesto que la falta de formas alternativas hace que los tri-gramas de las entidades sean los mismos que en A.

9.1.4. Experimento D. Ambos tipos de navegación de grafo

Como se muestra en el gráfico 8.1d, cuando se usan ambos tipos de navegación de grafo los resultados en todas las tasas mejoran respecto a los experimentos anteriores. La tasa de MB de primeros resultados crece hasta 94 % y la de perdidos desciende hasta 6 %. No aparece un sólo resultado intermedio. Para el caso de AOL, la tasa de primeros asciende a 80.95 % y la de perdidos desciende a 16.19 %, con una tasa de intermedios de of 2.85 %. No obstante, en esta ocasión si exploramos por qué aún hay resultados no reconocidos o clasificados en posiciones intermedias, nos encontramos con que los motivos ya no son los mismos para ambas fuentes.

- MB: Hay 6 resultados que aún no han clasificado en primera posición. Uno de ellos no pasó el corte de la función de bloqueo debido a que la canción está formada por un nombre muy corto de palabras comunes en el idioma castellano: “El amor”, de “Azúcar Moreno”. Los otros 5 resultados fueron piezas de música clásica que a pesar de haber superado el corte de bloqueo no fueron detectadas como suficientemente parecidas de acuerdo al criterio de Levenshtein. Un ejemplo ilustrativo de esto sería la consulta “Sonata for Piano no. 7 in C major, K. 284b/309: III. Rondeau. Allegretto grazioso”. Su equivalente en Discogs expresa la misma realidad, pero con convenciones de nombres distintas: “III - Rondeau. Allegretto Grazioso. Piano Sonata No. 7 C Major, K. 309 (284b)”. Estas cadenas son difícilmente reconciliables de acuerdo a la distancia de Levenshtein. Otros algoritmos orientados a tokens podrían ser más efectivos, pero también se da que algunas entradas detectadas en Discogs sí siguen casi la misma convención de nombrado a pesar de que expresen realidades distintas. La consulta que clasificó como primera para el ejemplo anterior fue “sonata no 2 in a major, op 100:

allegretto grazioso". La información sobre escritores en los casos de piezas clásicas no fue de tanta utilidad como en el resto, debido a que los autores son repetidos habitualmente (Mozart, Beethoven,...) y son asociados a pistas de nombres muy largos con apenas algunas de variaciones de caracteres entre ellos. Nuestra conclusión es que para piezas clásicas se deberían utilizar algoritmos más especializados en buscar contenidos clave en cadenas de texto. El diseño de MERA permitiría la inclusión de un algoritmo de este estilo en el flujo de ejecución.

- AOL: Los resultados en posiciones intermedias aparecen principalmente por haber detectado versiones de la canción buscada antes que la propia canción objetivo. La razón de los resultados no detectados ha sido, en todos los casos, que la canción objetivo no fue seleccionada entre las candidatas a hacer math en la etapa de bloqueo. Esto ocurre debido a erratas graves en nombres de canción o a nombres demasiado cortos y comunes.

9.1.5. Experimento E. Inclusión de la función de bloqueo adaptada a MERA

Cuando usamos ambos tipos de navegación de grafo además de la técnica de bloqueo descrita en la sección 7.2.3, todos los nodos objetivo en nuestras pruebas superan la fase de bloqueo. Como se muestra en el gráfico 8.1e, los resultados mejoran comparados con el experimento D para ambas fuentes, siendo la mejora más notable en el caso de AOL. En el caso de MB el único trabajo que en el experimento D no había superado la etapa de bloqueo sí la supera en el experimento E y clasifica en primer lugar, quedando la tasa de primeros resultados en 95 % y la de resultados perdidos en 5 %. Las piezas de música clásica ya habían superado el corte de bloqueo en el experimento D. Puesto que los algoritmos de reconciliación no se han visto afectados entre las pruebas D y E, siguen clasificando de la misma forma (posición inferior a 15). En el caso de AOL, la tasa de primeros resultados crece respecto al experimento D desde 80.95 % hasta 94.29 %, y la de resultados perdidos disminuye desde 16.9 % hasta 2.86 %. Los resultados en AOL que aún no han clasificado entre los 15 primeros se deben a erratas graves en el nombre de canción. Un ejemplo sería la entrada "ghostrider", que por el contexto hemos deducido que se refería a la canción "Ghost Writer". Para ambas fuentes, la tasa de primeros resultados en las condiciones del experimento E es superior

al 94

No obstante, esta mejora tiene una penalización en rendimiento. Hemos repetido los experimentos D y E 50 veces y la media de rendimiento es un 14.76 % peor cuando la función de bloqueo adaptada a MERA está activa. Pensamos que el rendimiento de esta función está fuertemente conectado a la naturaleza de los datos. Hemos medido cómo la función de bloqueo descrita en la sección 7.2.3 excede el número de candidatos a la salida de la etapa de bloqueo configurado en el experimento E. Esa cifra había sido fijada en 60, y hemos comprobado que la media de candidatos por consulta con la función activa ha sido de 68.12. Además, en el 94.62 % de los casos la consulta produjo exactamente 60 resultados. Sin embargo, en algunas ocasiones, hemos comprobado que la función devolvió una cantidad notablemente superior al límite fijado de 60. Esto ha ocurrido con consultas formadas por nombres cortos y palabras comunes. Durante la ejecución del experimento E, la consulta que ha producido un mayor número de candidatos validos ha sido "So what" de la artista "Ciara", con un total de 326 candidatos detectados

9.2. Efecto de la calidad de los datos

Como se muestra en los gráficos 8.1a, 8.1c, 8.1b y 8.1d, cuando usamos los dos tipos de navegación de grafo de MERA, nuestro prototipo arroja mejores resultados para la fuente a la que suponíamos mayor calidad de datos (MB) que para la fuente ruidosa (AOL). No obstante, cuando incluimos además la función de bloqueo especificada en la sección 8.2a, ambos tipos de fuentes presentan resultados de calidad muy similar, obteniendo AOL incluso una mejor tasa de resultados no detectados que MB (2.86 % vs 5 % de MB). No obstante, estos resultados pueden estar condicionados por el tamaño y naturaleza de las muestras, y sería necesaria la ejecución de más experimentos con distintas muestras y en diferentes condiciones para llegar a una conclusión con cierto grado de seguridad.

Capítulo 10

Descripción del Código

10.1. Estructura de paquetes

El código fuente entregado se estructura de la siguiente forma:

- Paquete **wmera**: Paquete principal de código, conteniendo los módulos correspondientes al reconciliador de entidades y módulos relacionados involucrados en el proceso.
    - Subpaquete *adapters*: contiene funciones para la conversión de datos entre distintas implementaciones de índices de q-gramas.
    - Subpaquete *controller*: contiene código para la ejecución de consultas y la devolución de resultados.
    - Subpaquete *graph_gen*: contiene código para la generación de grafos mediante el consumo de parsers generadores de objetos de modelo, además de una implementación del interfaz de grafo de MERA usando la librería de python RDFLib.
    - Subpaquete *infrastructure*: contiene interfaces e implementaciones distintas de índices de q-gramas para usar en la función de bloqueo.
    - Subpaquete *parsers*: contiene interfaces de parseadores que generan objetos del modelo así como implementaciones de los mismos para diferentes fuentes de datos.
    - Subpaquete *query_gen*: Contiene código para la generación de consultas en un modelo JSON a través del parseo de otro modelo de datos JSON.

  - Subpaquete *mera_core*: Contiene los principales paquetes del proyecto: modelo de clases, interfaces de sistemas a implementar, módulo de reconciliación de entidades y paquetes de comparación de cadenas.
  - Módulo *facade*: contiene código para consumir de forma sencilla el servicio de MERA desde la aplicación web que lo envuelve.
  - Módulo *factory*: contiene métodos que ocultan el proceso de construcción complejo de algunos objetos del sistema.
  - Módulo *utils*: distintos métodos de utilidades utilizados en diferentes puntos del código.
  - Módulo *word_utils*: distintos métodos de utilidades centrados en tratamiento de cadenas de texto.

- Paquete **apps**: Contiene diferentes scripts para la ejecución de algún tipo de aplicación, ya sea haciendo uso del paquete principal del proyecto donde se encuentran los módulos de reconocimiento de entidades o para la generación o análisis de datos para experimentos.

- Paquete **test**: Contiene los test implementados para el proyecto.

10.2. Diferencias entre prototipo WMERA y la especificación de MERA

El prototipo implementado no recoge todas las especificaciones de la arquitectura MERA, siendo las diferencias principales relativas a la cantidad de posibles tipos de datos manejados. Según el planteamiento de MERA, la especificación de tipos y asociaciones entre entidades debe ser absolutamente flexible, indicada a través de propiedades rdf o de contenido que una implementación de MERA pueda mapear a dichas propiedades para ser usadas en consultas contra un grafo.

Nuestra implementación, por contra, incorpora todas las técnicas propuestas de reconciliación de entidades en cuanto a flujo de ejecución, combinación de algoritmos y configuración de umbrales y relevancias, pero trabaja con un conjunto de tipos posibles de datos y propiedades cerrado, centrándose sobre todo en canción y artista (diferenciando personas y grupos). Esta

decisión tiene impacto sobre muchos puntos del sistema, siendo los más destacados:

- Rango de opciones en consultas.
- Interfaces entre módulos basados en un modelo común de objetos en lugar de en elementos rdf.

La otra diferencia fundamental con la especificación de MERA es que la relación entre canciones y artistas se establece mediante unión directa de sus nodos representativos en lugar de usar el esquema de nodos intermediarios descrito en la sección 7.3.3. Esta decisión se tomó sólo por cuestión de simplificar el modelo de cara al código y su mantenimiento, puesto que la empresa BMAT contratante a la que se entrega la aplicación no necesitaba mantener ni manipular la información de qué fuentes indican qué relaciones existen entre entidades principales.

En los posteriores apartados de esta sección profundizaremos en las diferencias nombradas.

10.3. Núcleo de MERA

10.3.1. Modelo común

La comunicación entre los distintos módulos de nuestro prototipo (parsers, reconciliador, generadores de consultas, formateadores de resultado,...) se realiza en general mediante el paso de parámetros de tipo simple o bien de objetos de un modelo de datos común. Esta decisión entra en conflicto con la definición de la arquitectura de MERA, pues el hecho de restringir las posibilidades a un conjunto cerrado de tipos de objetos impide que el usuario use el prototipo para reconciliar entidades que, aún teniendo que ver con el mundo de la música, no están recogidas en nuestro modelo de objetos. Con el trato homogéneo de entidades y relaciones mediante la especificación de elementos en rdf en un esquema de grafo coherente en la forma de relacionar la información la flexibilidad podría ser absoluta.

Las principales entidades representadas y recogidas en nuestro modelo de objetos son las de artista y canción, siendo artista una clase que especializan distintos tipos de artista (persona, grupo musical).

No obstante, imitando el esquema de grafo propuesto en el que se mantiene la fuente que proporciona cada pieza de información en el grafo, se ha implementado un modelo de datos al que hemos llamado "modelo etiquetado" en nuestro código que hereda del modelo simple en el que cada dato del objeto se asocia a la URI en el grafo que representa el dataset del que se obtuvo dicha información. Algunas capas del sistema operan con el modelo simple, mientras que otras hacen uso del modelo etiquetado.

El modelo etiquetado hereda del simple, respetando en todo caso el interfaz y la respuesta de las propiedades y métodos originales pero añadiendo nuevas propiedades que decoran las del modelo simple. Puesto que en el modelo etiquetado todas las clases heredan de su homóloga en el simple pero a su vez hay relaciones de herencia dentro del nuevo esquema, se ha hecho uso de mecanismos de herencia múltiple en python.

10.3.2. Matcher

El módulo que calcula la similitud entre una consulta recibida y los nodos de un grafo implementa por completo el algoritmo 1. No obstante, en el código entregado se usa sin excepción la función de bloqueo descrita en la sección 7.2.3. Además, el hecho de que el sistema trabaje con un modelo de objetos cerrado en lugar de sobre elementos rdf crudos hace que el interfaz de llamadas al parseador sea menos flexible que lo que plantea la arquitectura MERA. Los métodos de consulta que ofrece el matcher son o bien para encontrar una canción en el grafo o bien para encontrar un artista. También con motivo de que el sistema trabaja sobre un modelo de objetos cerrado, los posibles refinamientos para estas entidades se limitan a las propiedades de los objetos artista y canción recogidas en el grafo, o a la posible relación entre entidades.

A pesar de que el interfaz es capaz de recoger la mayoría de las propiedades de cada tipo de objeto, priorizando los intereses de la empresa contratante, los únicos refinamiento aplicados son los de entidades relacionadas, es decir, los de artistas asociados a canciones. El resto de refinamientos aportados en una consulta no tienen efecto sobre el proceso de reconciliación. Los demás refinamientos y posibilidades de búsqueda están pendientes de implementación. El momento en que se afronten estos requisitos puede depender de la priorización de tareas acordada con la empresa contratante.

10.3.3. Implementación de grafo

A pesar de que el matcher se comunica con una interfaz de grafo a la que hemos llamado MERAGraph, la única implementación de MERAGraph llevada a cabo ha sido un envoltorio para el un grafo creado con la librería de python RDFLib. La idea original era hacer una implementación de MERAGraph que estuviese basada en la comunicación con algún tipo de triplestore persistente. No obstante, pruebas de rendimiento al inicio del proyecto nos hicieron ver que un triplestore común, ya fuese por número de accesos a disco o por tráfico de red en el caso de alojarlo remotamente, introduciría una penalización de rendimiento inaceptablemente alta.

Hemos optado por una perdida de escalabilidad debido a la limitación de poder trabajar únicamente con grafos en memoria que persistimos a ficheros al finalizar los diferentes procesos a cambio de no tener la penalización de rendimiento que significaría involucrar un triplestore en el proceso. La implementación de una clase MERAGraph basada en un triplestore se plantea como trabajo futuro.

El rescate y guardado de datos en el grafo es llevado a cabo de cara a los módulos de generación de grafo y de linkado a través del consumo de la interfaz de MERAGraph, con un conjunto de órdenes cerrado que se refiere al rescate de datos de ciertas entidades en caso de consulta y una opción de introducir una tripleta con cualquier sujeto predicado y objeto en el caso de inserción. Internamente, el rescate de información del grafo se hace mediante consultas SPARQL parametrizadas con argumentos recibidos. Se usan un total de 16 consultas SPARQL parametrizadas distintas en nuestro prototipo para rescatar información del grafo.

10.3.4. Generador de grafo

A pesar de que durante el transcurso de este proyecto se ha experimentado con la generación de grafos representando distintas fuentes de datos, un único módulo de generación de grafo ha sido implementado. Dicho módulo se apoya sobre una o varias interfaces de parseadores de los que espera recibir objetos de datos de algunas de las clases del modelo común, delegando en la lógica del parseador la transformación de cada fuente de datos en estos objetos.

No obstante, la representación en forma de grafo de una fuente de información que originalmente no tiene estructura de grafo puede no ser un proceso trivial. Hemos probado con generadores de grafos funcionando prin-

cipalmente de tres formas distintas:

- *Modo aislado*: Cada objeto recibido por el parseador es considerado como único, es decir, se le asocia una identidad (una URI) propia. Sirve para procesos en los que podemos tener las seguridad de que el parser correspondiente enviará objetos que representan siempre entidades no repetidas y que además no tienen relación entre sí.

- *Basado en índices*: Si se esta convirtiendo en grafo una fuente de datos que contiene identificadores de entidades internos, esa información puede aprovecharse para resolver la identidad de los objetos recibidos. A cada identificador de objeto se le asocia una URI, y en lo sucesivo si se recibe de nuevo dicho identificador podemos resolver a qué entidad representa y establecer de forma correcta relaciones dentro del grafo.

- *Reconciliación sobre la marcha*: cuando la fuente parseada carece de identificadores internos y puede contener entradas potencialmente repetidas se ha probado a hacer una reconciliación sobre la marcha antes de incorporar en el grafo el objeto recibido por el parseador como una nueva entidad o como información que completa una ya existente. Al recibir un objeto se elabora un consulta que se lanza contra el grafo construido hasta ese momento. Si entre los resultados obtenidos hay alguno que supera cierta puntuación, se interpreta que el objeto ya está incluido en el grafo y se usa su información para añadir datos a cierto nodo entidad. De lo contrario, se crea un nodo entidad nuevo. No obstante, esta forma de generación de grafo tiene una gran penalización de rendimiento y no resulta segura.

Siempre que nos ha resultado posible hemos usado el generador funcionando en modo aislado o basado en índices. Las pruebas con el generador reconciliando entidades sobre la marcha no dieron resultados buenos ni en cuanto a rendimiento ni en cuanto a precisión. La mayor parte de entidades eran detectadas de forma correcta, pero rara vez todas, introduciendo asociaciones incorrectas o entidades duplicadas en el grafo.

Ontologías en el grafo

El principal objetivo de este trabajo era el planteamiento de un sistema eficaz para la reconciliación de entidades musicales, siendo el uso de grafos

rdf una herramienta que consideramos apropiada para lograr tal fin. No obstante, los grafos producidos durante el proceso de reconciliación podrían ser aprovechados de otra forma.

A la hora de elegir ontologías con las que representar los datos nos encontramos con que la mayor parte de conceptos que pretendíamos representar ya estaban definidos por otras ontologías de uso extendido, tales como RDF, RDFS, DC[1], o FOAF[2]. Encontramos de suma utilidad la ontología Music Ontology [58], que recoge numerosos conceptos referidos a entidades musicales. No obstante, nos encontramos con que el esquema de grafo propuesto por MERA tal y como se ha descrito en la sección 7.3.3 plantea un conflicto de compatibilidad con todas estas ontologías por cuestiones de rango en las propiedades. Por poner un ejemplo, la propiedad FOAF:name es definida para relacionar un nodo que representa una persona con un literal de tipo xsd:string que representa el nombre de dicha persona. MERA pretende darle ese uso pero, con el esquema de grafo propuesto en la sección 7.3.3, entre el nodo persona y el literal que representa su nombre situamos un nodo intermedio que une ambos conceptos y además se relaciona con la fuente de datos que proporcionó dicha información. Así, la propiedad FOAF:name no estaría uniendo una persona y un literal, sino una persona y un nodo auxiliar que apunta a dicho literal. De esta forma estamos incumpliendo la definición de la propiedad "name" en la ontología de FOAF.

La tarea de encontrar o definir ontologías que encajen con nuestro esquema de grafo propuesto para poder usar MERA para la fabricación de grafos reutilizables con datos de calidad de acuerdo a la definición de ontologías se plantea como trabajo futuro.

10.4. Código relacionado

Además de los módulos principales encargados de la resolución de consultas y la creación/consulta de grafos, han sido varios los paquetes de código desarrollados en el transcurso de este proyecto, bien para consumir los módulos principales, bien para nutrirlos de datos o bien para realizar algún tipo de control o función. En la próximas secciones llevaremos a cabo un repaso de los principales módulos extra implementados.

[1] Dublin Core
[2] Friend Of A Friend

10.4.1. Parseadores

Diferentes parseadores o parsers han sido implementados para fines distintos durante el transcurso del proyecto. Entendemos por parser aquella pieza de software que accede al contenido de un archivo, endpoint o alguna otra fuente de datos y transforma la entrada de forma que pueda ser consumida o leída por otro módulo. Los principales parsers desarrollados han sido:

- **Parser Discogs artistas**: Accede a un archivo XML[3] con información de artistas y los transforma en objetos Artist del modelo. El tamaño del archivo XML obliga a hacer una buena gestión de memoria, cargando sólo parte del fichero y devolviendo los objetos correspondientes bajo demanda.

- **Parser Discogs canciones**: Accede a un archivo XML con información de releases que contienen pistas. Recoge la información de cada pista y le añade la de la release que la contiene para generar un objeto Song del modelo. Como en el caso anterior, el tamaño del archivo obliga a hacer una buena gestión de memoria. Varias versiones de este parser han sido implementadas mediante herencia de un parser principal, para recoger unas u otras características de las canciones o para trabajar de forma conjunta con el parser de artistas.

- **Parser archivos BMAT**: BMAT tiene un formato propio de archivo conteniendo información musical. Se ha fabricado un parser que lo recorre fabricando objetos Artist y Song asociados. La cantidad potencial de datos obliga a hacer gestión de memoria.

- **Parser datos AOL**: En la fabricación de una muestra de consultas ruidosas para probar en los experimentos ha sido necesario explorar varias veces con distintos objetivos el log publicado por AOL en 2006. La cantidad de datos presente obliga también a gestionar adecuadamente la memoria.

- **Parser de experimentos**: Principalmente para la elaboración de datos para consultas, se han generado muchos archivos intermedios en diferentes etapas: claves, identificadores, consultas potenciales,... Para procesar cada uno de esos archivos ha sido necesario fabricar un parser a la medida de su formato y objetivo.

[3] eXtensible Markup Language

10.4.2. Índices de q-gramas

La función de bloqueo descrita en la sección 7.2.3 hace uso de una estructura de datos que contiene un índice de q-gramas. En la implementación de nuestro prototipo dicha estructura se divide en dos: una que hace las veces de contador de entidades en el grafo para el cálculo de valores de frecuencia de aparición inversa y otra que mantiene el propio índice de q-gramas para un tipo determinado de datos (artistas, canciones,...).

La función de bloqueo consume cierto interfaz de llamadas, que ha sido implementado por dos tipos de estructuras diferentes:

- *Índice en Mongo*: se ha creado un esquema de datos para el almacenamiento de información sobre q-gramas en diccionarios JSON para ser almacenado por una base de datos no relacional MongoDB.
- *Índice en memoria principal*: Se ha imitado en el esquema Mongo para crear una implementación de índice que mantiene todos los datos en memoria principal en objetos nativos de python que siguen la misma estructura de claves y valores esperados que las del esquema JSON diseñado para Mongo, facilitando así la conversión de los datos producidos por el índice Mongo y los almacenados en el índice en memoria principal.

La idea original para el índice de q-gramas fue la implementación del mismo en Mongo. No obstante, la consulta masiva de esta estructura durante la ejecución de la función de bloqueo descrita en el algoritmo 2 produce una gran penalización de rendimiento por la cantidad de accesos a disco, dando unos tiempo de ejecución difícilmente manejables cuando se tratan grandes volúmenes de datos.

Es por ello que se desarrolló una versión del índice de q-gramas que opera completamente en memoria principal. Dicho modulo mejora en torno a 100 veces el rendimiento de la versión Mongo, pero a costa de un gran consumo de memoria. empeorando la escalabilidad del prototipo.

10.4.3. Comprobadores de calidad de archivos

Durante el proceso de elaboración de ficheros para experimentos se han tenido que construir dos archivos de forma manual conteniendo datos para la elaboración de consultas a lanzar contra un grafo. Los archivos de consultas

para las fuentes MusicBrainz y AOL se encuentran adjuntos respectivamente en los anexos 13.1 y 13.2.

Ambos son ficheros de formato TSV[4] con numerosos campos y en los que ha sido necesaria mucha atención para no introducir o cambiar ningún dato. A fin de asegurar de forma automática tanto como sea posible la corrección estructural de tales archivos se han implementado un par de módulos que los recorren en busca de todos aquellos errores que puedan ser detectados de forma automática, resultando de notable utilidad a la hora de corregir incoherencias.

10.5. Pruebas

Todas las pruebas de código realizadas se encuentran en el paquete "test" del código fuente. Han sido diseñadas para ser ejecutadas con el framework *nose* de pruebas en python. Se han elaborado pruebas unitarias para ciertas clases, pero la mayor parte del tests son pruebas de integración en la que se comprueba el correcto funcionamiento de grandes regiones de código del prototipo.

Muchas de las pruebas requieren el acceso a una base de datos Mongo, que esperan encontrar en el puerto por defecto de Mongo (27017) en el equipo local. El script de arrancado de la base de datos en caso de contar con dicha instalación se ha adjuntado en el anexo 13.3. Algunas de las pruebas, además, precisan de la presencia de ciertos ficheros en una carpeta "files" situada en la raíz del código. No obstante, por el tamaño de dichos archivos en algunos casos y por el caracter privado de los mismos (relacionados con la empresa BMAT) en otros, no ha sido posible incluir esos archivos como parte del código de este proyecto. Por esta razón parte de las pruebas darán error al ejecutarse en un entorno que no cuente con esos elementos. No obstante, se ha adjuntado en la entrega del proyecto el código de todas las pruebas.

10.6. Modelo JSON de entrada de consultas

Se hace conveniente crear un formato común de entrada de datos a través del cual generar consultas con cierta configuración. Ha de existir un parser por cada nueva fuente de la que se pretendan extraer consultas, pero puede

[4]Tab Separated Values

resultar más sencillo, mantenible y desacoplado que los respectivos parsers transformen los datos originales a un modelo común en lugar de comunicarse con la interfaz del prototipo para la generación directa de consultas. En el anexo 13.4 se incluye un ejemplo de consultas y configuraciones en un modelo JSON diseñado para tal fin.

La estructura de dicho modelo JSON así como el significado y la obligatoriedad de sus claves se detallan a continuación:

- **config (opcional)**: Este campo sirve para especificar la configuración de MERA: algoritmos a usar en cada caso, umbrales, etc. Todo aquello no configurado en este nodo usara el valor por defecto de la implementación de MERA. Dentro del objeto config encontramos tres claves, también opcionales. En una especificaremos la configuración de ordenes (find_song, find_artist, find_genre,...), en otra las técnicas a aplicar para cada tipo concreto de dato (artista, canción, género, álbum,...)y en la restante bloqueos de candidatos en distintas fases.

  - **commands (opcional)**: contendrá sucesivas claves dentro de un rango conocido que se corresponderán con identificadores de órdenes. Para la orden de buscar canción, por ejemplo, será una clave "find_song". Dentro habrá dos claves, no obligatorias.
    - **threshold(opcional)**: número flotante positivo que indica que score mínimo tiene que arrojar un nodo al usar MERA para que sea tenido en cuenta en los resultados cuando usamos esta orden. Puede ser superior a 1 porque puede acumular los resultados de muchas comparaciones de campos concretos de score potencial 1.
    - **relevances(opcional)**: diccionario cuyas claves son valores de fields (ver posibles valores en el siguiente item de esta lista) apuntando a números flotantes en [0,1] que indican factor de relevancia del correspondiente tipo de elemento a la hora de hacer la búsqueda.
  - **fields(opcional)**: contendrá sucesivas claves que identificarán un tipo de datos de los que serán tratados (artista, álbum, canción,...), dentro de un rango de posibilidades conocido. Por ejemplo, para especificar el tipo de datos nombre de artista, la clave sería "artist". Contendrá dos claves, opcionales:

- **threshold(opcional)**: número flotante entre 0 y 1 que indica que score mínimo debe alcanzar una comparación de un par de cadenas concretas de este tipo de datos para ser tenida en cuenta en los resultados.
- **algors(opcional)**: nombres dentro de un rango conocido de algoritmos a usar para este tipo de datos en concreto. Se tratará de un nuevo diccionario en el que cada nombre es una clave a un nuevo diccionario de algoritmos, con recursividad potencialmente infinita. Esto se hace así porque algunos algoritmos necesitan (o admiten) ser configurados sobre otros algoritmos para funcionar. Levenshtein, por ejemplo, siempre debería aparecer asociado un diccionario vacío, porque no requiere de ningún otro algoritmo para funcionar. MongeElkan, en el otro extremo, debería tener siempre un diccionario con contenido, porque necesita otro algoritmo. Y algunos como podría ser Jaccard pueden o no llevar algoritmos asociados.

- **blocking(opcional)**: contendrá dos claves (también opcionales) que delimitan el número máximo posible de entidades a tener en cuenta en ciertos punto de ejecución del programa.
  - **blocking_function(opcional)**: limita el número máximo de candidatos que puede superar la fase de bloqueo para pasar a una comparación exhaustiva de la query con su nodo. El número de candidatos deberá ser menor o igual al entero positivo que aquí se incluya.
  - **result_query(opcional)**: limita el número máximo de resultados que una query puede arrojar. Si hay 80 entidades que superan las condiciones de umbral pero aquí se especifica que como máximo se devuelve 10, entonces solo las 10 con mejor score se devolverán.

- **queries(obligatorio)**: contendrá una lista con una sucesión de diccionarios, cada uno de los cuales representa un consulta. Dichos diccionarios query tendrá una serie de campos, siendo unos obligatorios y otros opcionales.

  - **type_of_query(obligatorio)**: indica que tipo de entidad estamos intentando buscar con esta query, o lo que es lo mismo, qué

orden estamos ejecutando. Contendrá un string dentro de un rango acotado de órdenes, tales como “find_song” o “find_artist”.

- **main_info(obligatorio)**: contendrá un string que identifica la cadena de texto principal de la query. Si la query es, por ejemplo, de tipo “find_song”, entonces este campo tendrá el título de la canción que tenemos como primer candidato. Si la query en cambio es un find_album, entonces tendremos el nombre del album.
- **extra_args(opcional)**: contendrá un diccionario almacenando todos los datos extra que se le pueden aportar a la query. Dicho diccionario contendrá claves dentro de un rango acotado que se corresponden con tipos de campos reconocidos (los potencialmente referidos en el fields de la configuración). Ejemplos sería “artist”, “song”, “album”,... Cada clave apuntará a una lista de strings crudos con la propia información. En caso de ser un campo que, de forma natural, no contenga lista (ejemplo: buscamos una canción y contamos con su fecha de lanzamiento, que tiene que ser única), se incluirá igualmente como una lista de un único elemento.
- **config(opcional)**: Su estructura es idéntica al config que aparece como clave en el diccionario raíz. Todos sus campos siguen siendo opcionales. De aparecer algún campo, sobrescribe la configuración a usar en el matcher sólo para esta query concreta.

Este modelo se ha elaborado pensando en ser válido para ampliaciones del proyecto. El prototipo entregado no hace uso de todas las configuraciones descritas.

10.7. Tratamiento de grandes volúmenes de datos

Han sido varios los puntos de este trabajo en los que ha sido necesario el procesamiento de grandes volúmenes de datos. Situaciones de ejemplo serían:

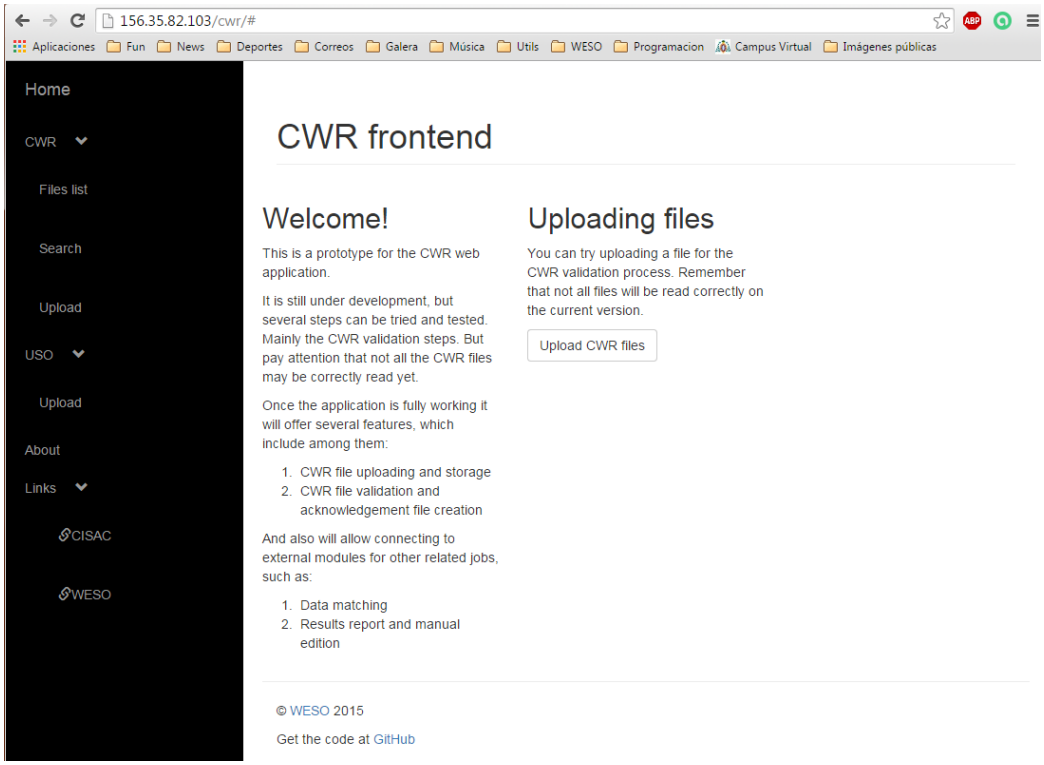
- Análisis de distintos tipos del log de búsqueda publicado por AOL en 2006, conteniendo más de 20.000.000 consultas.
- Parseo de ficheros de releases proporcionados por Discogs, con más de 45.000.000 pistas detectadas y un tamaño de archivo de más de 19 GB (contenido XML).

No contamos con un hardware de desarrollo capaz de procesar estos volúmenes de datos en memoria principal, por lo que ha sido necesario un estilo de programación en el que se hace una gestión medida del consumo de memoria. La tecnología más frecuentemente utilizada han sido los *generadores del lenguaje de programación python* y la *carga parcial de ficheros.*

Las sentencias generador pueden ser aplicadas sobre métodos de los que se espera obtener una lista de resultados. Cuando dichos métodos se programan usando generadores, en lugar de retornar una estructura de datos (array, lista, etc.) que contiene todos los elementos devuelven objeto que *genera* elementos bajo demanda del código llamador. Esto permite de forma potencial que procesamientos secuenciales de una sucesión de elementos no impliquen tener que cargar todos los elementos en memoria a un tiempo. Los generadores combinados con la carga parcial de ficheros nos permiten hacer una gestión de memoria en la que se mantiene en memoria principal sólo el contenido imprescindible para llevar a cabo una operación concreta.

Un ejemplo de esta técnica sería la creación de un grafo de canciones a partir de los contenidos del archivo de releases de Discogs, con un tamaño de más de 19 GB y más de 45.000.000 canciones. Para esta tarea se ha usado el método de parseo de ficheros XML *iterparse()* de la librería de python *xml.etree.cElementTree*. Dicho método funciona manteniendo un puntero a una posición fichero y un buffer de caracteres que recoge información hasta detectar determinado evento (fin/principio de etiqueta, aparición de propiedad,...) esto permite la extracción de nodos dentro de un xml correctamente formados sin comprobar la corrección del fichero XML al completo, además de mantener sólo en memoria principal el contenido en caracteres del buffer. El parser de este archivo se ha programado mediante el uso de generadores, por lo que el código que lo consuma en lugar de recibir una lista con tantos objetos canción como se encuentren en el archivo recibirá bajo demanda objetos canción de uno en uno. De la misma forma, el modulo para la creación de grafo se ha programado con el uso de generadores. Esto nos permite no tener que esperar a recibir todos los objetos canción para comenzar a producir tripletas rdf, sino introducir en el grafo las tripletas rdf correspondientes a un objeto canción, sacarlo de memoria y solicitar el siguiente al parseador. De esta forma hemos controlado el consumo de memoria a pesar de haber generado y gestionado en el proceso más de 45.000.000 de objetos canción.

Figura 10.1: Home de la aplicación web



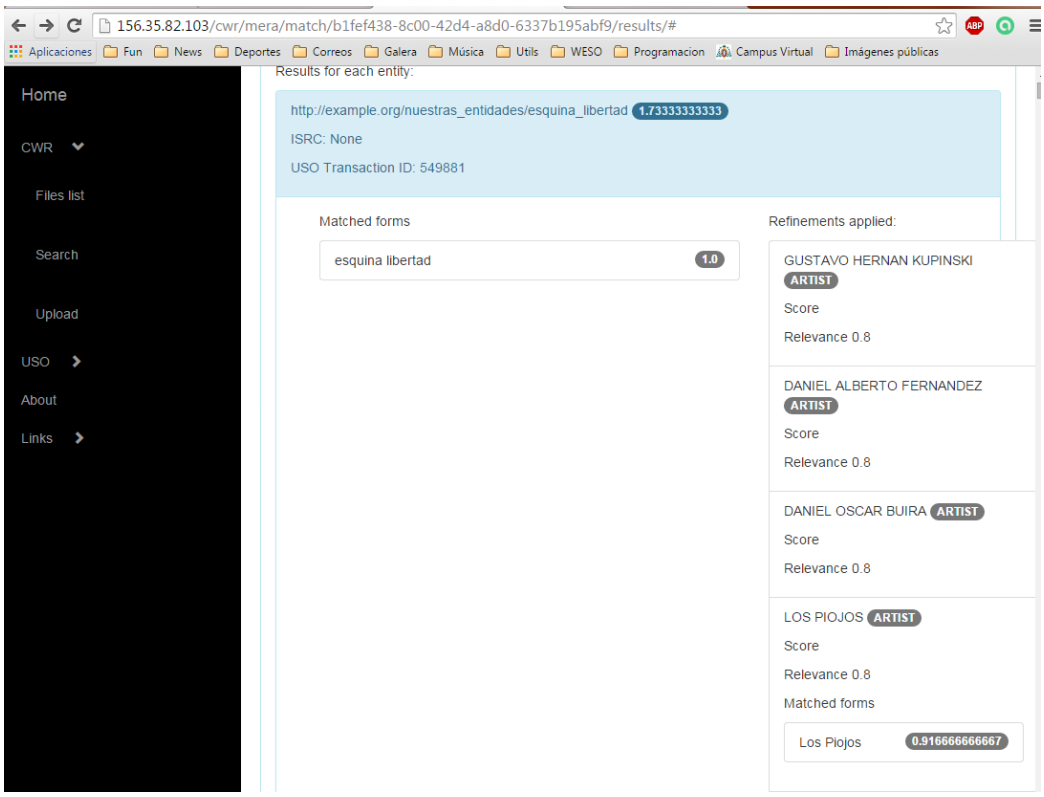
10.8. Aplicación consumidora del prototipo

En otro Trabajo Fin de Master llevado a cabo por el alumno Bernardo Martínez Garrido se desarrolla una interfaz web para el aprovechamiento del prototipo que hemos descrito. Dicha interfaz web permite la subida de archivos de distintas clases, invoca el proceso de reconciliación entre fuentes y muestra los resultados. En las figuras 10.1 y 10.2 se adjuntan capturas de pantalla de dicha aplicación.

10.9. Evolución del código

El código del prototipo descrito se encuentra en continua ampliación, corrección y evolución puesto que el proyecto con la empresa contratante aún no ha finalizado. El código evoluciona en un repositorio tipo git en la

Figura 10.2: Página de resultados del proceso de matching



plataforma BitBucket, pero por exigencias de la empresa contratante dicho repositorio ha de tener caracter privado. La lista de tareas abiertas en dicho repositorio (bugs/tareas pendientes) abierta en el momento de la elaboración de esta memoria es:

- Refactorización de paquete controlador
- Refactorización de strings hard-coded en paquetes generadores de consultas.
- Refactorización de estrategia de generación de tripletas en modo aislado.
- Refactorización del paquete de generación de grafos.
- Optimización de la interacción con base de datos Mongo mediante caché.
- Algunos algoritmos de comparación de cadenas parecen estar arrojando resultados excesivamente altos.
- Lectura de parámetros de configuración del modelo JSON de consultas.

Nuevos bugs son detectados y tratados según avanza el proyecto, de la misma forma que nuevas funciones pueden ser incorporadas o incluso excluidas o modificadas.

Capítulo 11

Presupuesto

En el fichero zip que se entrega con esta documentación, dentro de la carpeta "Presupuesto", se adjunto un fichero Excell en el que se muestra de forma tabular el presupuesto del proyecto, tanto interno como para el cliente, nombrado "71677368F_DFA_Presupuesto.xslx". Dicho fichero se estructura de la siguiente forma:

- **Presupuesto interno**:
  - Gastos procedentes de logística y materiales. Se incluye local, material de oficina, logística, hardware/software,... Cada elemento del presupuesto tiene una duración asignada que no tiene por qué ajustarse a la duración del proyecto (ejemplo: una licencia de software para 4 años). En esos casos se aplicará un porcentaje de reducción de gasto en base al tiempo de amortización estimado.
  - Gastos procedentes de salarios del personal contratado para llevar a cabo el proyecto. Cada empleado tendrá un sueldo base cuyo gasto será incrementado un 30 % para cubrir su seguro social. Se calcula el número de horas que cada empleado usará en cada sprint de la planificación. La plantilla descrita trata de ajustarse a la situación real:
    - Director de proyecto → Rol de investigador jefe 1.
    - Profesor que nos ha asesorado a menudo → Rol de investigador jefe 2.
    - Proyectante → Rol de becario investigador.

  - Resumen de totales: se incluyen cifras totales del coste de proyecto y cifras intermedias que han de ser usadas para los cálculos de prorrateo para el presupuesto del cliente. Se especifican además los valores de IVA[1] y el margen de ganancias del proyecto.

- **Presupuesto externo**:

  - Especifica la duración del proyecto, el número y rango de empleados involucrados, el número de horas que cada empleado empleará en el proyecto, total sin IVA y total con IVA.

Estrategia de prorrateo: en el presupuesto de cliente únicamente aparece coste de hora por empleado, número de horas, total sin IVA y total con IVA. La cantidad a prorratear se ha calculado con la siguiente formula:

$$€ \text{ a prorratear} = \text{coste sin IVA} - \text{coste empleados(no seg.social)} \quad (11.1)$$

El precio de cada hora de cada empleado se verá incrementado en la factura del cliente multiplicando su sueldo base por un factor de prorrateo (el mismo en todos los casos). Dicho factor es calculado mediante la siguiente fórmula:

$$\text{Factor aumento hora} = \frac{€ \text{ a prorratear} + \text{coste empleados (no seg.soc.)}}{\text{coste empleados (no seg.soc.)}} \quad (11.2)$$

Diferencia entre el presupuesto del cliente y el interno: El precio de la hora de trabajo de cada empleado se redondea a dos decimales para presentarlo al cliente y el resultado se usa para multiplicar por el número de horas trabajadas de cada uno. Por ello existe una diferencia nimia entre el resultado final del presupuesto interno y el de cliente (en torno a 2 € de diferencia en un precio final de casi 21.400 €).

[1] Impuesto sobre el Valor Añadido

Capítulo 12

Conclusiones y trabajo futuro

En el presente trabajo presentamos MERA, una arquitectura para un sistema de reconciliación de entidades musicales altamente configurable. Nuestro diseño introduce algunos nuevos conceptos/ideas sobre sistemas de reconciliación existentes, ya sean de propósito general o especializados en música:

- Aproximación basada en grafos que usa un esquema de representación de datos diseñado para etiquetar qué piezas de información proceden de qué fuentes de datos.
- Navegación de grafo configurable en la que se exploran formas alternativas representativas de cada tipo de nodo para ser usadas durante el proceso de reconciliación.

Hemos propuesto además una función de bloqueo basada en indexación de q-gramas y TF-IDF adaptada al esquema de grafo de MERA y al tratamiento de formas alternativas de una entidad. Hemos implementado un prototipo que recoge estas características y hemos comprobado que resulta efectivo en más del 94 % de los casos bajo las condiciones de nuestros experimentos, que incluyeron reconciliación de datos de fuentes con distinto nivel de calidad de datos.

Aún es necesario mucho trabajo para producir un sistema de reconciliación reutilizable que incluya todas las ideas especificadas en el presente artículo. No obstante, somos optimistas con los resultados obtenidos con nuestro prototipo y estamos actualmente trabajando en la implementación de ese sistema final, en la forma de un framework para el lenguaje de programación python. Estamos siguiendo las siguiente líneas de trabajo:

- Trabajo describiendo las formas apropiadas de construir el grafo que consume MERA.
- Pruebas profundas comparando una implementación de MERA con sistemas de reconciliación existentes, ya sean de propósito general o especializados en música.
- Prueba de las ideas de MERA en escenarios no relacionados con el mundo de la música.
- Incorporación de técnicas supervisadas a través de proporcionar al sistema datos de entrenamiento.
- Incorporación de mecanismos de procesamiento en paralelo.
- Mejora de la escalabilidad para almacenamiento y manejo de grafos.
- Mejora de eficiencia en acceso a grafos.
- Estudio de las ontologías existentes más apropiadas o desarrollo de una nueva para producir grafos reusables mediante MERA.

Bibliografía

[1] I. Amón and C. Jiménez. Funciones de similitud sobre cadenas de texto: una comparación basada en la naturaleza de los datos. In *CONF-IRM Proceedings*, volume 58, 2010.

[2] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 586–597. VLDB Endowment, 2002.

[3] M. Arrington. Aol proudly releases massive amounts of private data. *TechCrunch: http://www. techcrunch. com/2006/08/06/aol-proudly-releasesmassive-amounts-of-user-search-data*, 2006.

[4] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD*, volume 3, pages 25–27. Citeseer, 2003.

[5] T. R. Belin and D. B. Rubin. A method for calibrating false-match rates in record linkage. *Journal of the American Statistical Association*, 90(430):694–707, 1995.

[6] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal - The International Journal on Very Large Data Bases*, 18(1):255–276, 2009.

[7] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):5, 2007.

[8] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[9] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.

[10] M. Bilgic, L. Licamele, L. Getoor, and B. Shneiderman. D-dupe: An interactive tool for entity resolution in social networks. In *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, pages 43–50. IEEE, 2006.

[11] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 313–324. ACM, 2003.

[12] P. Christen. Febrl-: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1065–1068. ACM, 2008.

[13] P. Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Springer, 2012.

[14] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. *Kdd workshop on data cleaning and object consolidation*, 3:73–78, 2003.

[15] W. W. Cohen. A web-based information system that reasons with structured collections of text. In *Proceedings of the second international conference on Autonomous agents*, pages 400–407. ACM, 1998.

[16] W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems (TOIS)*, 18:288–321, 2000.

[17] A. Culotta and A. McCallum. Joint deduplication of multiple record types in relational data. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 257–258. ACM, 2005.

[18] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, Mar. 1964.

[19] H.-H. Do and E. Rahm. Coma: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 610–621. VLDB Endowment, 2002.

[20] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 85–96. ACM, 2005.

[21] U. Draisbach and F. Naumann. Dude: The duplicate detection toolkit. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2010.

[22] T. E. Dunning and B. D. Kindig. Determining a known character string equivalent to a query string, June 9 2009. US Patent 7,546,316.

[23] T. E. Dunning, B. D. Kindig, S. C. Joshlin, and C. P. Archibald. Associating and linking compact disc metadata, July 19 2011. US Patent 7,984,062.

[24] A. Elmagarmid, I. F. Ilyas, M. Ouzzani, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. Nadeef/er: Generic and interactive entity resolution. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1071–1074. ACM, 2014.

[25] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[26] G. A. Fink. *Markov models for pattern recognition: from theory to applications*. Springer Science & Business Media, 2014.

[27] W. R. Gilks. *Markov chain monte carlo*. Wiley Online Library, 2005.

[28] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, D. Srivastava, et al. Approximate string joins in a database (almost) for free. In *VLDB*, volume 1, pages 491–500, 2001.

[29] S. Guha, N. Koudas, A. Marathe, and D. Srivastava. Merging the results of approximate match operations. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 636–647. VLDB Endowment, 2004.

[30] P. A. Hall and G. R. Dowling. Approximate string matching. *ACM computing surveys (CSUR)*, 12(4):381–402, 1980.

[31] J. Hartnett. Discogs. com. *The Charleston Advisor*, 16(4):26–33, 2015.

[32] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *ACM SIGMOD Record*, volume 24, pages 127–138. ACM, 1995.

[33] T. N. Herzog, F. J. Scheuren, and W. E. Winkler. *Data quality and record linkage techniques*. Springer Science & Business Media, 2007.

[34] D. Holmes and M. C. McCabe. Improving precision and recall for soundex retrieval. In *Information Technology: Coding and Computing, 2002. Proceedings. International Conference on*, pages 22–26. IEEE, 2002.

[35] P. Jaccard. *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz, 1901.

[36] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[37] M. A. Jaro. Probabilistic linkage of large public health data files. *Statistics in medicine*, 14(5-7):491–498, 1995.

[38] P. Jokinen, J. Tarhio, and E. Ukkonen. A comparison of approximate string matching algorithms. *Software: Practice and Experience*, 26(12):1439–1458, 1996.

[39] P. Jurczyk, J. J. Lu, L. Xiong, J. D. Cragan, and A. Correa. Fril: A tool for comparative record linkage. In *AMIA annual symposium proceedings*, volume 2008, page 440. American Medical Informatics Association, 2008.

[40] D. V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems (TODS)*, 31(2):716–767, 2006.

[41] H. Kang, L. Getoor, B. Shneiderman, M. Bilgic, and L. Licamele. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(5):999–1014, 2008.

[42] D. E. Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[43] H.-C. Kum, A. Krishnamurthy, A. Machanavajjhala, M. K. Reiter, and S. Ahalt. Privacy preserving interactive record linkage (ppirl). *Journal of the American Medical Informatics Association*, 21(2):212–220, 2014.

[44] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[45] M. Larsen. Multiple imputation analysis of records linked using mixture models. In *Statistical Society of Canada Proceedings of the Survey Methods Section*, pages 65–71, 1999.

[46] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[47] B. Lynch and W. Arends. Selection of a surname encoding procedure for the statistical reporting service record linkage system. *Washington, DC: United States Department of Agriculture*, 1977.

[48] A. McCallum and B. Wellner. Object consolidation by graph partitioning with a conditionally-trained distance metric. In *KDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*. Citeseer, 2003.

[49] A. E. Monge, C. Elkan, et al. The field matching problem: Algorithms and applications. In *KDD*, pages 267–270, 1996.

[50] A. Mount. Allmusic. http://www. allmusic. com/. *Journal of the Society for American Music*, 7(03):359–361, 2013.

[51] G. Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[52] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970.

[53] H. B. Newcombe. *Handbook of record linkage: methods for health and statistical studies, administration, and business*. Oxford University Press, Inc., 1988.

[54] H. B. Newcombe and J. M. Kennedy. Record linkage: making maximum use of the discriminating power of identifying information. *Communications of the ACM*, 5(11):563–566, 1962.

[55] T. Peng, L. Li, and J. Kennedy. A comparison of techniques for name matching. *Journal on Computing (JoC)*, 2(1), 2014.

[56] L. Philips. The double metaphone search algorithm. *C/C++ users journal*, 18(6):38–43, 2000.

[57] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[58] Y. Raimond, S. A. Abdallah, M. B. Sandler, and F. Giasson. The music ontology. In *ISMIR*, pages 417–422. Citeseer, 2007.

[59] V. Rastogi, N. Dalvi, and M. Garofalakis. Large-scale collective entity matching. *Proceedings of the VLDB Endowment*, 4(4):208–218, 2011.

[60] G. Salton and M. J. McGill. Introduction to modern information retrieval. 1983.

[61] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278. ACM, 2002.

[62] M. Sariyar and A. Borg. The recordlinkage package: Detecting errors in data. *The R Journal*, 2(2):61–67, 2010.

[63] R. Schnell, T. Bachteler, and S. Bender. A toolbox for record linkage. *Austrian Journal of Statistics*, 33(1-2):125–133, 2004.

[64] R. Schnell, T. Bachteler, and J. Reiher. Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, 9(1):41, 2009.

[65] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[66] A. R. Stutzbach. Musicbrainz (review). *Notes*, 68(1):147–151, 2011.

[67] A. Swartz. Musicbrainz: A semantic web service. *Intelligent Systems, IEEE*, 17(1):76–77, 2002.

[68] J. R. Talburt. *Entity resolution and information quality*. Elsevier, 2011.

[69] S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems*, 26(8):607–633, 2001.

[70] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 350–359. ACM, 2002.

[71] D. Vatsalan, P. Christen, and V. S. Verykios. A taxonomy of privacy-preserving record linkage techniques. *Information Systems*, 38(6):946–969, 2013.

[72] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk-a link discovery framework for the web of data. *LDOW*, 538, 2009.

[73] W. Winkler. Issues with linking files and performing analyses on the merged files. *Proceedings of the Sections on Government Statistics and Social Statistics, American Statistical Association*, pages 262–265, 1999.

[74] W. E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. 1990.

[75] W. E. Winkler. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*. Citeseer, 1999.

[76] W. E. Winkler. Methods for record linkage and bayesian networks. Technical report, Technical report, Statistical Research Division, US Census Bureau, Washington, DC, 2002.

[77] W. E. Winkler. Overview of record linkage and current research directions. In *Bureau of the Census*. Citeseer, 2006.

[78] W. E. Yancey. Bigmatch: A program for extracting probable matches from a large file for record linkage. *Computing*, 1:1–8, 2002.

Capítulo 13

Anexos

13.1. Material para las consultas extraídas de MusicBrainz

El siguiente fichero TSV ha sido usado para la fabricación de consultas con datos de la fuente MusicBrainz:

```
discogsIndex  discogsID  nRelatedPersons  songName  Artists  Writers
166376  [r203207]7  4  New World Order  Mr. X & Mr. Y  Professor Klaus|Afrika Islam|WestBam
11468498  [r1829841]5  1  Colourless Colour  La Roux  -
27354677  [r367218]2  9  Corporal Jigsore Quandary  Carcass|The Berzerker|Vulgar Pigeons|Alienation Mental|Evocation  William Geoffrey "Bill" Steer|Jeffrey "Jeff" Walker|Ken Owen|Michael Amott
26663238  [r4397889]B5  4  Torpedo Girl  KISS|The Torpedo Girls  Vini Poncia|Ace Frehley
3452653  [r3582862]A2  4  Mandrake Root  Deep Purple  Rod Evans|Ritchie Blackmore|Jon Lord
21790270  [r2820034]A1  4  Green Onions  -  Steve Cropper|Lewie Steinberg|Booker T. Jones|Al Jackson, Jr.
38763481  [r2971636]B5  4  Island of Domination  Judas Priest  Rob Halford|Kenneth Downing|Glenn Tipton
9689000  [r4978894]1  9  Don't Let Go (Love)  En Vogue|Little Mix|Glennis Grace|Berget Lewis|Edsilia Rombley  Ivan Matias|Organized Noize|Andrea Martin|Marqueze Ethridge
12695014  [r2153095]A  4  You Don't Have to Say You Love Me  -  Vicki Wickham|Vito Pallavicini|Pino Donaggio|Simon Napier-Bell
13774423  [r504771]A  1  What I Want  Dead or Alive  -
5859010  [r933613]A  3  Rasputin  -  Fred Jay|George Reyam|Frank Farian
27709827  [r4619321]B  1  The Red Rooster  -  Willie Dixon
23385633  [r3942611]A  1  Up on Cripple Creek  -  Robbie Robertson
```

24857423 [r3411532]A5 2 Tenderly - Jack Lawrence|Walter Gross

34661041 [r1589237]A1 3 I Walk the Line Jeffrey Venturo|Blitzkid Johnny Cash

24123570 [r475934]A 1 metal guru Marc Bolan -

21518422 [r1992671]B3 2 Out of My Dreams - Richard Rodgers|Oscar Hammerstein II

20118776 [r892676]B1 1 Feel Like Makin'Love - Gene McDaniels

36510642 [r4648485]A4 2 It Could Happen to You - Jimmy Van Heusen|Johnny Burke

29077717 [r594938]A6 3 Molina Klaus Wunderlich|Creedence Clearwater Revival John Fogerty

13934081 [r2572322]A 1 Billie Jean - Michael Jackson

31327891 [r373288]1 1 Heart-Shaped Box - Kurt Cobain

21370578 [r2530956]F2 1 Weihnachts-Oratorium, BWV 248: Teil VI, LV. Recitativo "Da berief Herodes die Weisen heimlich" (Evangelista) - Johann Sebastian Bach

25530192 [r1343916]2-8 2 Lady Jane - Keith Richards|Mick Jagger

39288960 [r2151964]A 3 Eternally - John Turner|Geoffrey Parsons|Charlie Chaplin

24429831 [r1224903]B2 1 Symphony no. 7, "Sinfonia antartica": IV. Intermezzo: Andante sostenuto - Ralph Vaughan Williams

17738893 [r2720025]B3 3 Poet and Peasant: Overture Thomas Trotter Edwin Evans|Franz von Suppé

27205571 [r1328683]A 2 Living in America - Charlie Midnight|Dan Hartman

12648963 [r469591]1 5 Gonna Get Along (Without Ya Now) The Lemonheads|Mr. President|Viola Wills|She & Him Milton Kellem

32003081 [r1506936]B3 3 Storm in My Soul Gallagher & Lyle Graham Lyle|Benny Gallagher

12374587 [r6140303]A 4 You to Me Are Everything The Real Thing|Jadranko Ken Gold|Michael Denne

14907926 [r2246387]2 5 I See You (Theme from Avatar) Clare Teal|Leona Lewis Kuk Harrell|Simon Franglen|James Horner

29726788 [r1102208]A1 2 Special One Ultra Vivid Scene Kurt Ralske

25606337 [r2707175]2 1 Enter the Ninja Die Antwoord -

14850203 [r968857]B 4 Include Me in Your Life Diana Ross|Marvin Gaye Marilyn McLeod|Mel Bolton

1660011 [r240904]A 4 Holding On to Nothing Agnelli & Nelson Audrey Gallagher|Chris Agnelli|Robbie Nelson

14737825 [r577470]A 2 We've Only Just Begun - Paul Williams|Roger Nichols

35155007 [r874068]10 1 Kivi itkee vihreää Viikate -

387705 [r9115]A 3 Giv Me Luv Alcatraz Jean-Phillippe Aviance|Victor Imbres

22027365 [r1029095]A 3 Only When You Leave Spandau Ballet|Tony Hadley Gary Kemp

24996695 [r383660]2 4 Silence, and the Firmament Withdrew Dark Tranquillity Martin Henriksson|Niklas Sundin|Mikael Stanne

```
      39421691  [r545523]1  4  Main Title (Braveheart)  London Symphony Orchestra|Choir of Westminster
Abbey|The Cundeez  James Horner
      7285001  [r1177057]1  4  Toxic  -  Christian Karlsson|Pontus Winnberg|Cathy Dennis|Henrik
Jonback
      37556720  [r1064392]A  3  The Heart of Rock & Roll  Huey Lewis & The News  Huey Lewis|Johnny
Colla
      28591469  [r1337007]A  2  You Must Have Been a Beautiful Baby  -  Harry Warren|Johnny Mercer
      4980376  [r4392346]B4  4  Long Time Gone  Crosby, Stills & Nash|Crosby, Stills, Nash & Young|Deja
Blue Grass  David Crosby
      35498850  [r5140519]1  3  Learn to Be Still  Eagles  Stan Lynch|Don Henley
      14730715  [r431797]9  2  Not My Only Friend  The Teardrop Explodes  Julian Cope
      40582762  [r2736986]A  3  Go Where You Wanna Go  The 5th Dimension|The Mama's and the Papa's
John Phillips
      22902177  [r4923238]B5  1  Oktett: V. Fuge und drei altmodische Tänze (Walzer, Polka, Galopp)
Paul Hindemith  -
      7093523  [r5468850]3-6  1  Préludes, Livre I, L. 117: VI. Des pas sur la neige. Triste et
lent  -  Claude Debussy
      3342010  [r188472]B1  3  Let Her Go  Strawberry Switchblade  Rose McDowall|Jill Bryson
      9399807  [r1919049]A1  7  Le Plat Pays  Pierre Bachelet|Serge Lama|Nana Mouskouri|Les Enfoirés|Fernando
Lameirinhas|Koen De Cauter  Jacques Brel
      15499266  [r5590826]A  3  Da Ya Think I'm Sexy?  -  Rod Stewart|Carmine Appice|Duane Hitchings
      39337566  [r1025160]4  2  Dime  CAKE  John McCrea
      18943625  [r1456515]B1  6  Spinning Wheel  Blood, Sweat & Tears|Cæcilie Norby|Tom Wopat|Percy
Faith|His Orchestra & Chorus  David Clayton-Thomas
      25416870  [r3809223]2-3  4  It's Too Late Baby  Derek and the Dominos|Bob Dylan|The Crickets
Chuck Willis
      34224183  [r3628733]A  3  Unanswered Prayers  Garth Brooks|Pat Alger  Larry Bastain
      34843829  [r1338568]A2  2  There Is Nothing Like a Dame  -  Richard Rodgers|Oscar Hammerstein
II
      43108427  [r3169459]12  2  The Song Is You  -  Oscar Hammerstein II|Jerome Kern
      40031420  [r5969535]A2  9  In the Garden  Elvis Presley|Jim Reeves|Pat Boone|Over the Rhine|Sister
Rosetta Tharpe| The Anita Kerr Singers|James Roots Quintet|The Rosette Gospel Singers  Charles
Austin Miles
      16385257  [r1525414]A  2  Teach Me Tonight  -  Sammy Cahn|Gene de Paul
      27491574  [r1919340]CD16-6  6  The Inner Light  The Beatles|Jeff Lynne|Anoushka Shankar|Henrique
Cazes|Carlos Malta  George Harrison
      38832587  [r5637459]D1  1  Sinfonie Nr. 1 c-Moll, op. 68  -  Johannes Brahms
      11181737  [r759506]A5  1  Art Bitch  CSS  -
```

6380999 [r1016061]1 9 With Every Heartbeat Kleerup|Robyn|Girls Aloud|Beat Radio|The Futureheads|Grovesnor|Polarsets|Micedelia Andreas Kleerup

24976839 [r3637143]B3 2 Falling Like Rain Matt Skiba And The Sekrets Matt Skiba

14128656 [r1891641]A 1 Cholly (Funk Gettin'Ready to Roll) Funkadelic -

19151316 [r1737175]1AA 1 Mad World - Roland Orzabal

1167084 [r26755]A2 3 Emerge Fischerspooner Warren Fischer|Casey Spooner

15881010 [r467141]17 4 Straight to Hell - Mick Jones|Paul Simonon|Topper Headon|Joe Strummer

33854102 [r469777]A 1 See Emily Play - Syd Barrett

35670123 [r720168]A 2 Take That Look Off Your Face - Don Black|Andrew Lloyd Webber

28985787 [r3722988]B1 4 Nobody's Car Jethro Tull Ian Anderson|Peter-John Vettese|Martin Barre

31900596 [r1320317]A 2 (I Can't Get No) Satisfaction - Keith Richards|Mick Jagger

16478039 [r2632981]A 2 Got to Get You Into My Life - Paul McCartney|John Lennon

6916936 [r3452368]C 1 I See It in Us H2O -

8556480 [r6706963]6 1 El Amor Azúcar Moreno -

21106139 [r3151947]2 2 So Strong Katherine Ellis|Jason Herd -

15834545 [r5951608]B 1 Mon Peré (Für Meinen Vater) Mireille Mathieu -

38374264 [r416896]A 1 You're the One That I Want - John Farrar

13083337 [r2097835]8 1 Bioluminescent Neto -

29261947 [r4162056]B3 2 Sonata for Piano no. 7 in C major, K. 284b/309: III. Rondeau. Allegretto grazioso Daniel Barenboim Wolfgang Amadeus Mozart

13129130 [r2143921]3 1 Heroine The Dogs DÁmour -

34136753 [r4307077]17 3 Sacrae Cantiones Liber secundus: IX. O sacrum convivium Vocalconsort Berlin|James Wood Carlo Gesualdo

31436364 [r737200]A 3 Lass mich nie mehr allein Peter Alexander Kurt Feltz|Heinz Gietz

6832795 [r427290]14 1 So träum ich Selig -

41097869 [r763077]B 2 You Just Like Me 'Cos I'm Good in Bed Skyhooks|Missy Higgins -

6674971 [r5759659]14 1 A Bad Case of Melancholy The Spent Poets -

30408893 [r2206955]C1 2 The World Today Canned Heat|John Lee Hooker -

17128191 [r2640406]1 2 Ragnaroek 2014 Rossomahaar Lazar

39564906 [r404828]B5 3 Manon Serge Gainsbourg|Lulu|Bambou -

31466014 [r2568185]2 1 Baby Blues Love Unlimited Orchestra -

5506501 [r959811]A 1 C'mon and Swim Ray Columbus & The Invaders -

7997426 [r2205374]1-1 1 Pêche à la mouche Django Reinhardt -

30137913 [r4331192]7 1 Upside Down(Album version) Denki Groove -

8778365 [r1502645]A5 1 Not Cool The dB's -

28765200 [r4335988]B3 1 Theme & Variations #2 Bill Holman -

7477862 [r2960745]A1 1 It's My Thang MC Luscious -

7297869 [r1278580]B 1 Dream of Dreams Joe Sample -

13.2. Material para las consultas extraídas de AOL

El siguiente fichero TSV ha sido usado para la fabricación de consultas con datos de la fuente AOL:

file number of line number of artists original query song artist1 artist2 ... Found/not found Comment RELEASE ID

w 48 1 the video whats left of me by nick lachey whats left of me nick lachey V - [r1408386]1

w 63 1 the shake neil the shake neil V - [r4677857]12

w 130 1 money by o jays money o jays V s@for the love of money [r6858722]A

w 133 1 lyrics i wish by omarion i wish omarion V - [r1357788]1

w 206 1 jack ingram lyris where ever you are where ever you are jack ingram V - [r5208742]1

w 263 1 queen songs fat bottom girls fat bottom girls queen V - [r1280281]2

w 311 1 fine young canibals band drives me crazy drives me crazy fine young canibals V s@she dirves me crazy [r319519]1-1

w 340 1 im in love with a stripper video by t-pain im in love with a stripper t-pain V - [r6259958]2

w 344 1 nine inch nails lyrics everyday is exactly the same every day is exactly the same nine inch nails V - [r656547]001

w 458 2 get low by lil john and the eastsideboys featuring the ying yang twins get low lil john and the eastsideboys the ying yang twins V Puede haber relacion de productor [r2081450]1

w 490 1 u2 - pride in the name of love pride in the name of love u2 V - [r396573]A

w 502 2 down rakim y ken y video down rakim ken y V - [r3764184]1

w 554 1 free ridin'dirty master p download ridin'dirty master p V d@living legend certified d-boy [r3696937]11

w 729 1 hit em up lil wayne lyrics hit em up lil wayne V - [r570373]12

w 766 1 lyrics - aerosmith - don't want to miss a thing don't want to miss a thing aerosmith V - [r1898882]1

w 773 1 gouda e40 gouda e40 V - [r861216]A5

w 807 1 warm bed video; jamie foxx warm bed jamie foxx V - [r588811]1

w 813 1 lyrics wateva by remy martin wateva remy martin V s@whuteva [r1141490]A1

w 906 1 uncut version of beep by the pussycat dolls beep the pussycat dolls V - [r649066]A1

w 928 1 neyo - when your mad when your mad neyo V s@when you're mad [r3934255]1

w 947 1 when i get you alone by robin thicke when i get you alone robin thicke V - [r812252]1

w 951 1 lick shots by missy elliot the song lick shots missy elliot V - [r199842]A1

w 1034 1 lyrics to what hurts the most by rascak flatts what hurts the most rascak flatts V - [r2345841]1

w 1049 1 butterfly album mariah carey lyrics butterfly mariah carey V - [r455168]2

w 1059 1 walk- pantera walk pantera V - [r3139704]1

w 1200 1 im rightlyrics touch it remix by busta rhymes touch it remix busta rhymes V Many remixes [r1571326]2

w 1212 1 cadillac don peanut butter jelly lyrics peanut butter jelly cadillac don V - [r2116288]1

w 1218 1 lyrics sean paul - temperature temperature sean paul V - [r3827750]A

w 1259 1 doing too much by paula deanda doing too much paula deanda V - [r856160]1

w 1282 1 areosmiths i dont wanna miss a thing song in only instrumental i dont wanna miss a thing in only instrumental areosmiths V No hay una version puramente instrumental [r1898882]1

w 1350 1 chamillionare lyrics to riding dirty riding dirty chamillionare V s@ridin'(explicit) [r655796]A2

w 1356 1 i miss you aaliyah i miss you aaliyah V s@miss you [r480474]1-15

w 1360 1 michelle williams sun will shine again lyrics sun will shine again michelle williams V - [r2951114]1

w 1374 1 between me and you ja rule lyrics between me and you ja rule V - [r3187404]1

w 1395 1 lyrics to it mustve been love by roxette it mustve been love roxette V s@it must have been love [r1532797]1

w 1436 1 best friends by 50cent best friends 50cent V - [r6781318]2-6

w 1527 2 so what-field mobb ciara so what field mobb ciara V - [r3207813]A3

w 1531 1 listen to hold me now by kirk franklin hold me now kirk franklin V - [r230436]9

w 1543 1 lyrics to the song my best friend by tim magraw my best friend tim magraw V - [r1896419]5

w 1612 1 will smith lyrics and fresh prince of bel air theme lyrics fresh prince of bel air theme will smith V - [r1285614]B2

w 1630 1 whuteva lyrics remy ma whuteva remy ma V - [r1141490]A1

w 1674 1 lyrics sos by rihanna sos rihanna V - [r633599]A1

w 1775 1 lyrics to you re my best friend by queen you re my best friend queen V - [r1295705]A

w 1849 2 down remix by rakim y keny down remix rakim keny V - [r3764184]1

w 1857 1 so what lyrics by jazze pha so what jazze pha V jazze pha is producer [r3207813]A3

```
w 1884 1 red hot chili peppers venice queen lyrics venice queen red hot chili peppers
V - [r368516]16
w 1893 1 janet jackson i miss you much i miss you much janet jackson V s@miss you
much [r1007075]A
w 1921 1 listen to miley cyrus the song this is the life this is the life miley cyrus
V a@Hannah Montana [r1068959]08.
w 1935 1 watch black buddafly bad girl video bad girl black buddafly V - [r5609399]A2
w 2037 1 rob base joy and pain joy and pain rob base V - [r4286466]A
w 2050 1 lyrics to sweet lady by tyrese sweet lady tyrese V - [r1416112]4
w 2054 1 that so raven theme song lyrics that so raven raven V - [r3543770]14
w 2065 1 lyrics to my way by frank sinatra my way frank sinatra V - [r6603022]B1
w 2087 1 lyrics to rihanna sos rescue me sos rescue me rihanna V Distinto de sos
a secas, creo [r2730783]A2
w 2092 1 black flag lyrics nervous breakdown nervous breakdown black flag V - [r371255]A
w 2095 1 i miss someone by johnny cash songfacts i miss someone johnny cash V s@i
still miss someone [r1355401]B1
w 2156 2 gimme that lyrics by chris brown ft lil wayne gimme that chris brown lil
wayne V Distinto de gimme that a secas [r1316745]1
w 2164 1 ghost rider rjd2 ghost rider rjd2 V s@ghostwriter [r1672434]6
w 2184 1 friggs lyrics bad word for a good thing bad word for a good thing friggs V
- [r4020235]A1
w 2189 1 listen to do what it do -jamie foxx do what it do jamie foxx V - [r588811]11
w 2206 2 lyrics for busta rhymes and papoose address me as mister address me as mister
busta rhymes papoose V - [r1650769]A1
w 2224 1 this love lyrics pantera this love pantera V - [r2865846]6
w 2229 1 suffix lil wayne mixtape suffix lil wayne V - [r4667199]15
w 2261 2 biggie and jay z i love teh doug lyrics i love teh doug biggie jay z V -
[r2277432]11
w 2311 1 aero smith lyrics don't want to miss a thing don't want to miss a thing aero
smith V - [r1898882]1
w 2413 1 lyrics for so what by ciara so what ciara V - [r3207813]A3
w 2508 2 nu-jersey devil and the game- face of la track face of la nu-jersey devil
the game V - [r2078535]2
w 2536 1 so what- feild mob lyrics so what field mob V - [r3207813]A3
w 2552 1 no more by 3lw lyrics no more 3lw V - [r3717871]11
w 2557 1 i'm a believer christina milian i'm a believer christina milian V d@Be Cool
- Original Motion Picture Soundtrack|s@Believer [r4900198]9
```

w 2569 1 listen lil jons snap your fingers snap your fingers lil jons V s@snap yo fingers [r1389350]A1

w 2591 1 lyrics for we be burnin by sean paul we be burnin sean paul V - [r525486]1

w 2596 1 love keisha cole love keisha cole V - [r5044579]2

w 2600 1 natasha bedingfield size matters lyrics size matters natasha bedingfield V - [r660900]12

w 2603 1 eric church how bout you how bout you eric church V - [r4399810]3

w 2665 1 pussycats buttons lyrics buttons pussycats V - [r778663]1

w 2701 1 i want it all queen free clip i want it all queen V - [r3646106]1

w 2748 1 i'm in love with a stripper remix f twista i'm in love with a stripper remix twista V He is a featurer [r2805717]5

w 2769 1 infatuated memphis bleek music codes infatuated memphis bleek V - [r6238647]1

k 323234 2 paula deanda ft. baby bash doing too much doing too much paula deanda baby bash V d@Full Tilt Remix Volume Four [r856160]1

k 323253 1 martina mcbride broken wing karaoke download broken wing martina mcbride V s@a broken wing [r5967679]5

k 323269 1 children of the corn - american dream american dream children of the corn V - [r1704642]B4

k 323271 4 listen to lil wayne fat joe paul wall rick ross holla at me baby holla at me baby lil wayne fat joe paul wall rick ross V d@Listennn - The Album (Clean) [r1382202]6

k 323273 3 http www.easy-sharing.com 334085 17 hustler's prayer - notorious b.i.g. feat. jim jones produced by green lantern .mp3.html hustler's prayer notorious b.i.g. jim jones green lantern V d@Alive On Arrival [r4157341]17

k 323292 3 shawna ft. jim jones and ludacris - getting some remix getting some remix shawna jim jones ludacris V s@Getting Some Head (Remix)|d@ [r2243069]9

k 323316 2 free nelly furtado featuring timbaland promiscuous download for itunes promiscuous nelly furtado timbaland V - [r776143]B1

k 323352 2 lyrics to field mob song so what featuring ciara so what field mob ciara V - [r3207813]A3

k 323395 3 01-busta rhymes -i love my bitch feat will i am and kelis www.pctrecords.com .mp3 i love my bitch 01-busta rhymes will i am kelis V d@Kick Push [RMX] / I Love My Bitch [RMX] [r5233842]B

k 323399 2 webjay - paula deanda feat baby bash - doing too much doing too much paula deanda baby bash V - [r856160]1

k 323400 2 'touch it'remix feat. lloyd banks and papoose video 'touch it'remix lloyd banks papoose V - [r1307988]A2

k 323414 2 bossy - kelis feat. too short lyrics bossy kelis too short V - [r1939897]1

k 323487 3 music video for i never snitch remix by scarface featuring the game and beaniesiegel i never snitch remix scarface the game beaniesiegel V d@My Homies Part 2|s@never snitch [r942045]1-3

k 323497 2 i'm in love with a stripper remix f twista akon lyics i'm in love with a stripper remix twista akon V Sin akon... [r2805717]5

k 323506 2 lyrics for paula deanda's doin to much feat baby bash doin to much paula deanda baby bash V - [r856160]1

k 323615 1 lyrics to bestfriend remixby 50 cent bestfriend 50 cent V - [r6781318]2-6

k 323651 2 ashlee simpson l.o.v.e lyrics for remix featuring missy elliot l.o.v.e ashlee simpson missy elliot V - [r3256663]4

k 323690 2 about you song featuring william lyrics by mary j blige about you mary j blige william V - [r785454]B3

k 323699 2 timberland & magoo feat fat man scoop - you got served soundtrack - now drop now drop timberland & magoo fat man scoop V s@drop [r4528251]2

k 323749 2 rompe remix by daddy yankee feat. g-unit rompe daddy yankee g-unit V d@Reggaeton Club Bangerz Vol. 20 [r6483289]A1

k 323764 2 dynamite hack- anyway featuring emily kate anyway dynamite hack emily kate V - [r5510577]42

k 323825 3 lil jon feat. e-40 & sean paul - snap yo fingers lyric snap yo fingers lil jon e-40 sean paul V d@Snap Yo Fingers [r1389350]A1

k 323656 6 http t-pain f.twista paul wall pimp c mjg too short a-kon im in luv wit a stripper lyrics im in luv wit a stripper t-pain paul wall pimp c mjg too short a-kon V - [r2805717]5

k 323940 3 holla at me baby lyrics paul wall lil wayne rick ross holla at me baby paul wall lil wayne rick ross V d@Listennn - The Album (Clean) [r1382202]6

k 323975 4 touch it remix with dmx papoose rah digga missy elliot lyrics touch it remix dmx papoose rah digga missy elliot V d@Touch It (Remixes) [r1307988]B1

k 324041 3 so what lyrics by jazze pha feat. ciara and field mob so what jazze pha ciara mob V jazze pha is producer [r3207813]A3

13.3. Script ejecución Mongo

Las siguiente ordenes de shell deben ser ejecutadas en un sitema con una instalación de mongo en el puerto por defecto (27017) para la ejecución de las pruebas:

```
cd {$PATH_INSTALACION_MONGO}
./mongod.exe -dbpath ./data/db
```

13.4. Ejemplo de consultas para MERA en JSON

{ “config” : { “blocking”: { “blocking_function” : 40, “result_query” : 5 }, “commands” : { “find_song” : { “threshold” : 1.60, “relevances” : { “artist” : 0.70, “album” : 0.80 } } “find_artist” : { “threshold” : 1.70, “relevances” : { “song” : 0.90, “album” : 0.50 } } }, “fields”: { “song” : { “threshold” : 0.70, “algors” : { “Levenstein” : {}, “MongeElkan”:{ “Levenshtein”:{}, “DamerauLevenshtein”:{} } } }, “artist” : { “threshold” : 0.90, “algors” : { “MiAlgortForArtist” : {}, “NeedlemanWunsch”:{}, “MongeElkan”:{ “NeedlemanWunsch”:{}, “MiAlgortForArtist”:{} } } }, “album”:{ “threshold” : 0.90, “algors” : { “Levenshtein”:{} } } } }, “queries”: [{ “type_of_query” : “find_song”, “main_info” : “Bam Bam”, “extra_args” : { “artist” : [“Ariana Grande”, “Nicki Minaj”, “Another One ”], “album” : [“The title of the album”] } }, { “type_of_query” : “find_artist”, “main_info” : “Linkin Park”, “extra_args” : { “song” : [“In the End”, “Crawling”, “Castle of Glass”] } }, { “type_of_query” : “find_song”, “main_info” : ”Bailando”, “extra_args” : { “artist” : [“E. Iglesias”] } “config” : { “commands” : { “find_song” : { “threshold” : 1.60 } }, “fields”: { “song” : { “threshold” : 0.90, “algors” : { “Levenstein” : {}, “MongeElkan”:{ “Levenshtein”:{}, “DamerauLevenshtein”:{} } } }, “artist” : { “threshold” : 0.70 }, “album”:{ “threshold” : 0.90, “algors” : { “Levenshtein”:{} } } } }] }

13.5. Copia del artículo de investigación

Elsevier Editorial System(tm) for Expert Systems With Applications
Manuscript Draft

Manuscript Number:

Title: MERA: Musical Entities Reconciliation Architecture

Article Type: Review Article

Keywords: Record linkage; Music metadata reconciliation; Adaptable architecture; Rdf graph

Corresponding Author: Mr. Daniel Fernández-Álvarez, Jr.

Corresponding Author's Institution: University of Oviedo

First Author: Daniel Fernández-Álvarez, Jr.

Order of Authors: Daniel Fernández-Álvarez, Jr.; Jose Emilio Labra Gayo, PhD

Cover Letter

Apr 30, 2015

Dear Editor:

Please find the submission of our manuscript entitled "***MERA: Musical Entities Reconciliation Architecture***" for your kind review. This paper proposes an adaptable architecture of a system of record linkage, specially designed to cover issues related with metadata of musical entities. We also implemented a prototype and include some experiments in the paper trying to validate our ideas. I confirm that this submission is original, has not been published elsewhere, and is not currently under review elsewhere and all authors have agreed to the manuscript submission.

I would very much appreciate if you consider the manuscript for publication in **The Journal of Expert Systems with Applications**. Please let me know if you need any more information to complete the submission process.

Look forward to your consideration.
Sincerely yours,

Daniel Fernández Álvarez

MERA: Musical Entities Reconciliation Architecture

D. Fernández-Álvarez[a,1,*], J.E. Labra Gayo[a,1], D. Gayo-Avello[a,1]

[a] *St. Valdés Salas, 33007, Oviedo, Spain*

Abstract

The problem of entity recognition or record linkage has been largely studied despite of the fact that it is still an opened issue. The proliferation of large databases with potentially repeated records across the World Wide Web drives into a generalized interest to find precise and efficient methods to detect duplicated entries when no reliable unique identifiers are available. This is desirable both to detect duplicated records within a database and to recognize as pointers to the same real-world object two records in different databases.

There is a set of issues that may appear when facing a record linkage scenario that are potentially shared by databases of different nature, such as the presence of misspellings. However, the different types of sources may present different types of particular issues. We have studied the cases of datasets containing information linked to the music world, finding that is hard to elaborate a list of problems common to every musical database. Data quality, naming conventions and content nature may affect the list of conciliation tasks to face in each case. We also find that some musical concepts are usually named with different but valid forms that identify the same reality. An illustrative example is the way to designate an artist or group, where artistic name, civil name(s), group members or alias could be provided.

In this paper we are describing MERA (Musical Entities Reconciliation Architecture), an architecture thought to link two databases adapting the techniques and reconciliation algorithms to each particular case. MERA also includes mechanisms to manage third party sources to improve the results. Our approach is based in web semantic technologies, storing and organizing the information in rdf graphs. MERA defines a graph schema to organize the information, not necessary linked to particular ontologies, and propose a way to explore graphs adapted to that schema. We have implemented and successfully evaluated a prototype that catches most of MERA 's design aspects.

Keywords:
Record linkage, Music metadata reconciliation, Adaptable architecture, Rdf graph

*Corresponding author

Email addresses: UO212626@uniovi.es (D. Fernández-Álvarez), labra@uniovi.es (J.E. Labra Gayo), dani@uniovi.es (D. Gayo-Avello)

[1] WESO Research Group, University of Oviedo

1. Introduction

In this paper, we have put effort in providing an improvement for the specific task of joining records of databases containing metadata linked to the music world. For that purpose, we have designed MERA (Musical Entities Reconciliation Architecture), an architecture of a system to discover links between elements of different databases that represent the same real world entity. MERA is able to adapt the linking process of the records to the different content and nature of each database, trying to reach rates of precision and recall as high as possible.

Examples of fields usually contained in musical databases are titles, artist names, albums, genres, etc. The task of recognizing this kind of content is strongly connected to the widely studied problem of record linkage, since it consists in the detection of records or entries referring to the same real-world entity. However, we have designed MERA with the assumption that the type of metadata linked to the music world presents a certain number of peculiarities that should be considered. For instance, there are many specific cases of correct forms, or at least recognizable forms, in which we could express the name of an artist, including but not limited to:

- *Artistic names vs civil names*: "Stefani Joanne Angelina Germanotta" vs "Lady Gaga".
- *Naming conventions*:"The Beatles" or "Beatles, the".
- *Official or widely extended alias*: "The King of Rock" instead of "Elvis Presley".
- *Mixings between civil names and artistic names*: "Shakira", "Shakira Isabel Mebarack Ripoll", "Shakira Mebarack",...
- *Acronyms*: "System of a down" or "SOAD".
- *Usual misspellings*: "Bruce Springsting" instead of "Bruce Springsteen".
- *Name of an artist linked to a song that should actually be linked to a group*: "Michael Jackson" instead of "The Jackson Five".

Issues such as misspellings or acronyms are not specific of the music world metadata, but they can be found in databases or datasets of different nature. However, issues such as the existence of both artistic and civil name are exclusive of *artists*' specification. By contrast, when trying to conciliate other types of musical entities, e.g., *songs*, a different set of particular problems related to the nature of songs may appear. An example could be the management of the word "feat" (or variations such as "ft.", "featuring", etc.), When "feat" appears in a song title, it usually means that in that title there is a name of a collaborator

included. Both "feat" and its following words may be discarded of the song name itself. However, they can possibly be computed in some other way since they can become useful information.

In noisy or hand-made databases it is also possible to find extra words at the beginning or at the end of a song title, that are for instance linked to the radio program in which a song was played or to the place of a live performance. This may become even more troublesome in specially noisy databases such as those formed by the compilation of standalone audio files' metadata. When handling audio files wrongly labelled, it is possible to find titles that in fact contain all the associated metadata (artist, date, genre...).

Another example of a musical concept that presents associated issues due to its special nature is *genre*. When dealing with genres, it could happen that the same song is specified as *pop* in a database, as *rock* in a second one and as *pop-rock* in a third one. Sometimes, the same genre is even named with different forms that are in fact expressing the same reality.

Our assumption is that finding general reconciliation rules between two particular databases is far from being trivial, as well as finding appropriate rules or strategies to conciliate each field of those databases. The result could drastically change if it is compared to the rules that may be used when handling a different pair of sources. Trying to establish general rules could drive into an unnecessary number of failures (false positives/negatives)when identifying two records of different databases as forms of the same real entity. The inference of reconciliation rules in a particular case through the use of training data may be handy for covering issues such as misspellings, naming conventions or even noisy prefixes/ suffixes [74], but they will not be enough when trying to link entities expressed with strings that do not have enough common characters (example: "The King of Rock" should be recognized as "Elvis Presley").

The system we have designed tries to adapt to all those scenarios through the use of graph concepts and web semantic technologies. Our approach consists in turning the information of one of the target databases into a custom rdf graph G containing all the information(name variations, alias, common misspellings, ...) of every database record as well as the relations between those records. The records of the second database are turned into complex queries that will be launched against G. The result of each query would be the list of the most similar nodes to the target record according to:

- String distance based functions.
- Use of all the alternative identifying forms of a concept.
- Graph navigation in order to detect shared associated entities for disambiguation purposes.

For improving the results, G may be enriched with extra sources. If we have to conciliate two sources A and B, where A contains civil names of artists and B contains artistic names, but we also count with a source C that includes both civil and artistic names, then it will be possible to turn B records in a graph

G and link records of C with nodes of G. The obtained results would allow us to build an enriched graph G' containing all B records in nodes decorated with information of C. Then making queries with entities of A against G' will potentially improve the result of making queries against G.

The system may be able to use different reconciliation algorithms for each pair of databases, and even for each field of those databases, trying to cover all the issues linked to the nature of the data. Our solution, at least potentially, will be able to reach better results with more prior knowledge of the data issues, since the user is the agent that should specify the algorithms to use. MERA allows configuration about the different properties that should be considered, the reconciliation algorithms to apply in each case and the threshold of similarity that a result must reach to be accepted. It would also provide mechanisms to incorporate ad-hoc algorithms in the reconciliation process.

In section 2 we will make an overview over existing techniques and systems of record linkage. In section 3 we will explain in detail MERA architecture. In section 4 we present a prototype implementing most of MERA architecture and the results of some experiments. In section 5 we expose our conclusions and future work.

2. State of the art

Definition. Record linkage, also referred as object identification [67, 68], data-cleaning [18], aproximate matching or aproximate join [27, 28], fuzzy matching [2] or entity resolution [6], is used to identify pieces of information that refer to the same reality when no unique identifiers are available. A basic use case could be the detection of duplicate entries within a file or the detection of equivalents across two databases.

Research work about record linkage has been largely based on Fellegi and Sunter seminal paper of 1969 [24], inspired in the ideas introduced by Newcombe [53]. Record linkage is presented as a classification problem, where an entity pair can be classified as "matching" or "non-matching". Fellegi and Sunter propose using largely unsupervised methods for this task, carefully establishing the probabilistic foundation of record linkage theory. Since that, several scientific communities have adopted the proposed schema to formulate the problem in its own way, producing many reusable techniques and technologies to solve it [75, 13].

String comparators. Since record linkage becomes a problem due to the lack of unique reliable identifiers, traditional approaches are highly based in string comparators [75, 13]. It is frequent when facing a matching problem not being able to conciliate entities through an exact (character-by-character) match, due to typographical errors, naming conventions and some other issues. In fact, being able to recognize different strings that represent the same real-object has been, and still is, a major research project in computer science [29, 50, 13]. There has been many work to define string similarity measures [48, 50, 14, 11] and to provide methods to match individuals using them [35, 36, 5, 73, 44]. Some

work was also done with technichs based in knowledge-intensive approaches for the database environment [31, 67, 68]. Phonetic encoding algorithms have also been used to face this task, being most of the word focused in English talkers [41, 33, 46], but also adaptations for other languages have been provided [55].

Training data. Deciding which is the most accurate strategy to apply in order to get the best possible results is not a trivial task [9]. Several works indicate that there is not such an algorithm or combination that can outperform all the rest in terms of accuracy and efficiency [75, 37, 1, 14], not even if we try to compare specific subsets of algorithms such as string-edit distance metrics [54]. Some research lines have put effort in the design of methods to automatically detect which algorithm or combination of algorithms from a known set of possibilities works better for a particular scenario. This can be done through providing training data human-built containing a set of pairs qualified as "matching" or "non-matching". Several researches showed how machine-learning could be applied in record linkage situations where is possible to provide training data, even if it is a small amount [59, 74]. Early approaches used Hidden Markov models [56] for the learning process [9, 8], and nowadays it is still a widely extended strategy, but different techniques have been proposed [25], such as Markov Chain Monte Carlo or Conditional Random Fields[43, 47, 26, 17].

Collective matching. The techniques used to determine the similarity between two records through applying string distance metrics over their attributes are known as *FBS*[2]. There are occasions in which FBS are not enough to properly determine the similarity between two entities, specially when a disambiguation is needed [39]. In those scenarios checking relations between entities in addition to entities' features may be a mechanism to improve matching.

Traditional FBS approaches match each individual independently. By contrast, approaches in which relations between records are considered to produce a result are known as *collective entity resolution* [7]. The representation of relations between entities fit well in graph structures, so these kind of approaches are usually graph-based [13]. Different ideas have been developed, such as building relationship graphs where entities are represented by nodes and relations (possibly weighted) are edges [39, 7] or using dependency graphs, where a node represents a similarity between a pair of entities, which has edges if it is depending upon a similarity of another pair [19]. Those graph-based approaches combined with FBS can outperform the matching quality of FBS standalone[19, 39, 7]. However, collective approaches usually carry a lack of scalability over FBS systems [57]. Nevertheless, it has been shown that those algorithms could be adapted to be more scalable [57].

Reducing the number of comparisons. Data matching presents some other challenges different to the pure string recognition, such as efficiency or scalability of

[2]Feature-Based Similarities

matching systems when dealing with huge amounts of information. If we are trying to merge two sets of elements $A = \{a_0, a_1, \ldots, a_n\}$ and $B = \{b_0, b_1, \ldots, b_m\}$ it is highly desirable not having to compare every entity a_i in A with every entity b_j in B. This becomes a crucial point when the sets A and B are, for example, commercial databases with millions or billions of entries. The reduction in the number of potential matches to consider is called *blocking*. The first significant approach to avoid unnecessary pair comparisons was made for databases that stored personal information [52]. The idea was to consider as potential matching pairs only those records that agreed on a characteristic such as surname or date-of-birth. Since that, many approaches have been proposed, e.g., sorting of records in order to keep similar contents together [31], clustering of candidates with computationally cheap functions before employing more expensive methods comparing pairs [11] or q-gram indexing [4].

Existing systems. Developing or designing a reconciliation system may need to include most of the reviewed algorithms and techniques: different string comparator algorithms, supervised methods using training data, blocking functions, collective matching, etc. In addition, some extra challenges or decisions must be addressed, such as accepted input/output formats, mechanisms to interact with the user (API, library, web application,...), configuration options or the possibility of including extra algorithms/work flows.

It is hard to make an overview of existing commercial systems, mainly for two reasons. Firstly, it is hard to judge when a system is "a reconciliation system" and when the reconciliation process is just a part of a bigger product. Secondly, those systems rarely public their technical details [13]. If we have an inside look at open source projects or researching prototypes, we can find several examples of general purpose (not specialized in the music world or any further context) matching systems. Some significant examples of those systems could be Dude [20], D-Dupe [10, 40], SILK [70], BigMatch[76], FEBRL[12], FRIL [38], Merge ToolBox[61, 62], OYSTER[66], RRecordLinkage[60], WHIRL[15], NADEEF/ER[23] or PPRIL [42].

All those systems have been proved to effectively tackle the challenge of entity matching. However, they are heterogeneous systems that could be classified and compared from different points of view:

- Main capabilities: Detection of duplicates within a dataset, conciliation of several datasets, or both tasks.
- Interaction with the user: user-friendly graphic interfaces, libraries, function packages,...
- Purpose: Solving a particular reconciliation problem, being used as benchmarks of entity recognition techniques,...
- Degree of flexibility: possibility to choose techniques, algorithms or workflow to apply during the matching process, and/or possibility to implement new modules to work with the original system.

- Machine learning techniques to use with provided training data.
- FBS or collective matching.
- Availability of blocking mechanisms.
- Availability of other features such as *privacy preserving matching*, handy when several organizations take part in the matching process but the information to be linked is sensible or should be encrypted[69].

Regarding approaches specialized in reconciliation of musical content, we have dug into recent published patents of systems designed for merging and managing music metadata finding some results [22, 21]. However, these works are hardly comparable to MERA. One of them [22] is an invention to let the user know as much metadata of a CD as possible [22], and it introduces many concepts out of our scope, such as management of a databased composed by users' introduced data, the computation of vectors composed by duration of tracks or the reconciliation of tracks through audio fingerprints. Nonetheless, despite the fact that using third party or intermediary files to improve the linking process is not a recent idea [71], this patent is the only one among the explored systems that specifically includes in its design the management and also the trustworthiness of several sources. The other work [21] purposes a technique based in q-grams that can be potentially applied in different contexts, but whose effectiveness has been checked against musical entities. However, that technique is presented as an strategy, not as a system, and it would be feasible to implement it and to include it in MERA's work-flow.

An example of open-source system specialized in music reconciliation is MusicBrainz Picard [64]. However, this systems, as one of the described patents [22], uses record linkage as a part of a bigger application. In addition, the systems is specially designed to match content of audio files with the entries in MusicBrainz database, not to be used with any pair of sources.

To summarize, there are several valid general-purpose or music-centred systems for the task of entity recognition, but none of them proposes some of the points included in MERA's design:

- Graph schema designed to store information of several sources, maintaining which pieces of information have been provided by which source(s).
- Graph navigation strategy adapted to that graph schema in which, given a certain node n_α that represents an entity, different alternative forms of that entity and different related nodes may be found.

3. System Architecture

Overview. MERA defines an strategy to combine several categories of algorithms or heuristics in order to conciliate two sets of entities $A = \{a_0, a_1 \dots a_n\}$ and $B = \{b_0, b_1 \dots b_m\}$, where the entities of B are stored in a rdf graph G. Each record a_i is processed independently to find its most similar records in

B. The user of MERA should be allowed to define a set of rules in order to determine certain conditions that an entity b_j must fulfil to be considered as a possible match. Consequently, every entity a_i could be associated to none, just one or several results, depending on the number of entities $b_j \in B$ that fit in the conditions. In case of having more than one result for a same entity a_i, those will be presented ranked, being firstly positioned those entities detected as more similar.

If we think of MERA as a black box, these would be the expected inputs and the received outputs:

- *Input1*: set of entities $A = \{a_0, a_1 \dots a_n\}$, for which every a_i will be processed independently.
- *Input2*: rdf graph containing the set of entities $B = \{b_0, b_1 \dots b_m\}$.
- *Input3*: configuration data, in which all MERA's options may be specified.
- *Output*: Association of each element a_i with all the elements in B that fit in defined matching conditions. In case of having several results for an entity a_i, they will be presented sorted in decreasing order by degree of similarity with a_i.

Some of the configuration parameters of Input3 should necessary be specified by the user, but most of them should have a default value. Those possible configurations include, not being limited to: algorithms to apply for each kind of node, a top number of entities in B to potentially associate with an entity in A, a minimum similarity score for each type of entity... We will go deeper in configuration possibilities in section 3.4 later in this article.

Using extra datasets. Despite the fact that we have described MERA as an algorithm that links elements of a set A with similar entities in a set B, more sets (datasets) can be used in the process. Let's say that we have got a set A containing civil names of artists and a set B containing artistic names. If we try to link elements of A and B, independently of the applied algorithms, we may only have success when the civil and the artistic name are the same (ex: "Jennifer Lopez") or when, at least, they share some tokens ("Shakira" vs "Shakira Isabel Mebarack Ripoll"). However, in these cases in which those names do not share common significant sequences at all (ex: "Stefani Joanne Angelina Germanotta" vs "Lady Gaga"), we could not be able to detect a match using string similarities. We assume that using extra datasets that let us to collect different alternative forms of en entity may improve the matching results. Let's say that we are counting with a third dataset C in which the artist entities include both civil and artistic names, so it can be used as a "bridge" to link entities of A and B through transitivity. Being a_α a record in A, b_α a record in B ,c_α a record in C and denoting a link between two entities u and v as $u \Leftrightarrow v$, if $a_\alpha \Leftrightarrow c_\alpha$ and $b_\alpha \Leftrightarrow c_\alpha$, then $a_\alpha \Leftrightarrow b_\alpha$.

The way to make use of that extra sources would be by using MERA to match those new sources with B and adding the matched data to Graph G

obtaining G'. Later, we would use MERA to link A and G'. There are some public datasets of music metadata that may be used for this purpose, such as discogs [30], allmusic [49] or musicbrainz [65].

Query structure. Records in A are turned into queries that may involve a single or several entities at a time. A query in MERA context must specify a main type or main order o, a main content m, and a set R of an indefinite number of typed refinements r, which would be pairs $r_i = (t_i, s_i)$.

Let's consider an scenario in which we have got a set A containing artists that can be linked with song titles. One of the artists we can find in A is A_α ="Mike Chang", with no associated songs. Another artist is A_β ="Kurt Hummel", and he has associated the song A_γ ="For Good". Now, let's consider that we want to conciliate the artist of A with a graph G. For that, we must turn entities in A into queries. The resulting query $q_\alpha = (o_\alpha, m_\alpha, R_\alpha = \{r_\alpha *\})$ for A_α would contain o_α = "artist", m_α ="Mike Chang" and $r_\alpha = \{\lambda\}$, being $\{\lambda\}$ an empty set. The resulting query $q_\beta = (o_\beta, m_\beta, R_\beta = \{r_\beta *\})$ for A_β would contain o_β ="artist", m_β ="Kurt Hummel" and $R_\beta = \{$("song","For Good")$\}$. If A_β had more songs associated, then more pairs $r_{\beta_i} =$ ("song", s_{β_i}) would be added to R_β. If we knew that Kurt Hummel was born in USA, we could also add a pair such as ("country","USA") to R_β

If we wanted to link songs instead of artists, then the resulting query $q_\gamma = (o_\gamma, m_\gamma, R_\gamma)$ for the song A_γ would contain o_γ ="song", m_γ ="For Good" and $R_\gamma = \{$("artist", "Kurt Hummel")$\}$.

As it can be seen through the previous examples, a MERA query q_i searches for a main raw string m_i associated to certain type o_i. And also, all the refinements or extra data provided for disambiguation purposes should be also labelled with a type. That is, MERA follows an strategy in which every piece of information must necessary be associated to a single type. The algorithm is not thought to look for raw and unlabelled pieces of strings in every possible node of Graph G.

Parallel computing. Every query generated from the entities of A is processed independently of each other against G. In addition, G is accessed in a read-only way. With this scenario, we think that a feasible option to include parallel computing mechanisms in a MERA implementation could be by processing each query in different threads/machine nodes, as well as using a system to store G able to dispatch several asynchronous petitions at a time. However, the pure total/partial replication of G in not synchronized copies may work. There is no need to include synchronizing mechanisms between the different copies of G since they are not supposed to be mutated. For the same reason, non-blocking I/O can be used, since the structure serving G content will receive read-only petitions.

3.1. MERA's adaptable algorithm

Query execution. In algorithm 1, we have described in pseudo-code the MERA algorithm from the moment in which a Query q is received to the moment

Algorithm 1 MERA Query

Require: Graph G
Require: Query q
Require: Config C

▷ Blocking stage

candidateNodes ← G.nodes
for all f_{BLK} ∈ C.blockingFunctions **do**
    candidateNodes ← f_{BLK}(q, candidateNodes)

▷ Main comparison

tmpResults ← [λ]
for all aNode ∈ candidateNodes **do**
    formScores ← {λ}
    for all aForm ∈ aNode.alternativeForms **do**
        formScores ← formScores + f_{CMP}(q.mainContent, aForm , q.type)
    if max(formScores) ≥ C.minMainScore(q.order) **then**
        tmpResults[aNode] = max(formScores)

▷ Refinements

for all aRefType ∈ q.refinementTypes **do**
    for all aCandidateRes ∈ tmpResults **do**
        candidateRefs ← G.getRelated(aCandidateRes.node, aRefType)
        for all aRefinement ∈ q.refinementsOfType(aRefType) **do**
            formScores ← {λ}
            for all aForm ∈ candidateRefs.alternativeForms **do**
                formScores ← formScores + f_{CMP}(aRef, aForm, aRefType)
            if max(formScores) ≥ C.minRefScore(q.order, aRefType) **then**
                refScore ← max(formScores) · C.relevance(q.order, aRefType)
                tmpResults[aNode] ← tmpResults[aNode]+ refScore

▷ Filtering and sorting results

results ← [λ]
k ← C.maxResults(q.order)
for all candidate ∈ tmpResults **do**
    if candidate.score < C. minScore(q.order) **then**
        results← results + candidate
sort(results)
if |results| ≤ k **then**
    return results
else
    return results[0: k]

in which MERA returns its associated entities in Graph G. However, some functions, methods, properties, and structures need to be clarified in order to fully understand the algorithm's behaviour:

- Property *nodes*: invoked over a Graph G, it returns a set containing all the graph nodes.
- Property *blockingFunctions*: it contains a set of blocking functions defined by the user.
- Functions f_{BLK}. Each function f_{BLK} expects a query q and a set of nodes S, and it returns a subset S' of S in which only the best candidates to match with q in S according to f_{BLK} criteria can be found.
- Structure $\leftarrow \{\lambda\}$: represents an empty set.
- Structure $[\lambda]$: represents an empty dictionary, in which pairs (key $\rightarrow$ value) are stored.
- Property *alternativeForms*: it contains all the string alternative forms of the node over which it is invoked.
- function f_{CMP}: it expects two strings and a type in order to select the appropriate comparator algorithms. It returns a similarity score in the range $[0, 1]$.
- Property *mainContent*: it contains the main string of a given Query q.
- Property *type*: it contains the main type of a Query q, i.e., the type of entity that should be linked with the main string in q,.
- Method *minMainScore*: invoked over a Config C object, it receives a type of order and it returns the minimum score associated whit the main entity in that order.
- Property *order*: it contains the order of a Query q.
- Property *refinementTypes*: it contains a set of types of refinements associated to a Query q.
- method *getRelated*: invoked over a Graph G, it expects a node n and a type t. It returns all the nodes related with n through an edge of type t.
- Method *refinementsOfType*: invoked over a Query q, it expects to receive a Type t and it returns a set with all the specific refinements associated with q of type t.
- Method *minRefScore*: invoked over a Config C, it expects a type of order o and a type of entity t, and returns the minimum score that a entity of type t should reach to be accepted when executing an order of type o.

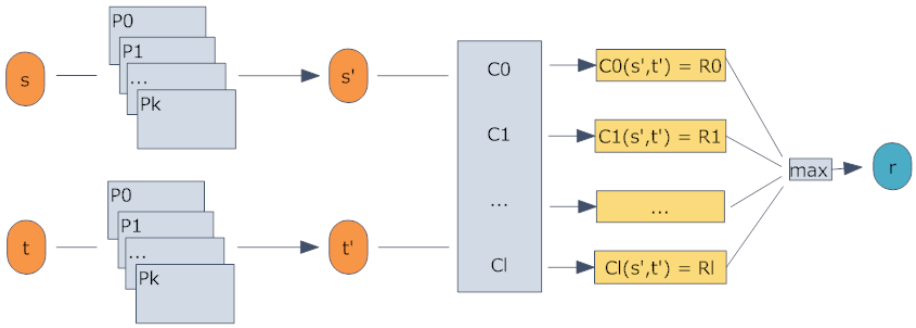
- Method *relevance*: invoked over a Config C, it expects a type of order o and a type of entity t, and it returns the relevance factor of a refinement of type t when executing an order of type o.
- Method *maxResults*: invoked over a Config C, it expects a type of order o and returns the maximum number of results when executing an order of type o.
- Function *sort*: it expects a set or dictionary S and it modifies the order of the elements in S, sorting them in descending order.

We have distinguished four parts in algorithm 1 (as shown in the algorithm's comments). This division helps us to explain MERA's algorithm in a more informal way as the succession of the following four stages:

- *Blocking stage.* Config object C stores a set of blocking functions to filter candidates of Graph G. All those functions are applied over G nodes in order to obtain a subset of nodes to be compared with q.
- *Main comparison.* Query q has a main type o and a main string content m. m is compared with all the alternative forms of the candidates obtained in *step 1*.
- *Refinements.* Query q may have a set of extra data or *refinements*. For each refinement, G is navigated starting from the candidate nodes of *step 1*. If coincidences that are good enough according to C are found, the obtained score is added weighted according to parameters in Config C to the respective candidate node score.
- *Filtering and sorting results.* The results associated to a query q are sorted in decreasing order. The user is allowed to define a maximum quantity k of results to be associated to a query q. If more results than k were found, only the k better would be returned. Otherwise, all the results would be returned.

String comparison. In lines 8 and 17 of algorithm 1 we use a macro f_{CMP} to represent the process in which the similarity between two strings s and t is obtained by using several comparison algorithms. f_{CMP} also receives a type o, expected to be the type of the entities that s and t represent. o must be specified for a potential accuracy profit, since different algorithms may be applied to conciliate s and t regarding their type o. To justify this, let's say that we want to link two datasets A and B, both of them containing data about song names and release dates. An attempt to standardize a song name may require to eliminate redundant blanks, lower-casing strings, etc. Whereas, standardizing dates may require to detect the date format of s and t and to turn s and t to a common format if needed. For comparison tasks, song names may include string-edit distances or token-based approaches. However, dates should probably be parsed in order to calculate a numeric difference in some kind of time unit.

Figure 1: Mapping two strings s and t into a real number r



When mapping s and t to their degree of similarity r , once the set $P = \{P_0, P_1, \ldots, P_k\}$ of pre-precessing techniques and the set $C = \{C_0, C_1, \ldots, C_l\}$ of comparison techniques has been selected based on the type o of s and t, the same work-flow is followed in all cases, graphically described in figure 1. s is turned into s' and t into t' pipping all algorithms P_i in P. That is, the input of P_0 would be s and, in general terms, the input of P_i would be the output of P_{i-1}. The final output s' would be the output of P_k. Then, for each C_i in C, we would obtain every $R_i = C_i(s', t')$. The greater value of $\{R_0, R_1, \ldots, R_i\}$ would be taken as the result r, having $f_{CMP}(s, t, o) = r$.

3.2. Methods

MERA defines a strategy to combine three different categories of functions, including pre-processing, blocking, and string comparison. Each set of functions will be used together so as to reach a result for every matching query, i.e., for every attempt to find similar entities in a set B for en entity a_i in a set A. The user should be able to specify several functions of each category to be used in an execution of a MERA implementation. In the following sections we will make an overview of each category functions, as well as explaining how the system combines them or behave if the user does not specify any.

3.2.1. Pre-processing

A typical issue to solve when comparing text from different sources (or even from a single source with not consistent data patterns) is the need of applying some transformations or heuristics in order to standardize data[32]. We are referring to issues such as lower/upper case, unnecessary white spaces or the presence of punctuation marks. However, there could be more aspects to be standardised than the cited examples, and even the cited examples could be wrong examples, in some contexts, or at least not so trivial as it could look like. For instance, when dealing with acronyms, the decision of removing, replacing or preserving may have an important impact on the result. E.g.: The acronym “B.o.B.” of “Bobby Ray Simons” could be turned into “BoB”, possibly matching

with other people called Bob in the available data, or into "B o B", sequence that may be hard to identify as an acronym.

And example of more complex standardization would be the attempt of removing meaningless words, such as prepositions or conjunctions. This may be an easy task when dealing with entities in a single idiom, but problematic or not so trivial in sets of entries containing data in several languages.

Different techniques should be selected regarding to the data content, quality and naming conventions. The rest of employed algorithms of different categories (comparison, blocking) may also be a factor to bear in mind when selecting the set of pre-processing functions.

All the algorithms or heuristics of pre-processing categories should expect to receive a string and should return an equivalent one. That output should be the form used in blocking and comparison stages. All the techniques will be applied sequentially to every target text. If no algorithms of preprocessing were specified by the user, the text in blocking and comparison stages should be processed in the same form that it appeared in the original set.

3.2.2. String comparison

When trying to detect equivalent records in two sources with no shared unique identifiers, most of the linking process relies into string similarity measures [13]. For instance, if we are dealing with databases storing personal data, it might be necessary to check if semi-identifying fields such as name, surname or birth date coincide or are similar in order to identify records that potentially refer to the same real-world person.

We can define similarity functions as functions that map two strings s and t to a real number r, where higher values of r mean greater similarity between s and t [14]. The resulting real number r is usually a value in the range [0,1], where 1 means equality (according to the algorithm criteria) and 0 means total inequality. All the similarity algorithms integrated in MERA are expected to work like that, mapping two strings s and t to an r value in the range [0,1], independently of the criteria applied to measure the degree of similarity.

There are several types of algorithms thought to measure similarity between two strings. We will make a brief overview of them in the following sections.

Character-based. Character based distances, also called string-edit distances [50], consider the strings s and t to compare as an ordered succession of characters. They generally base their results in the number of deletions, additions substitutions and, in some instances, transpositions necessary to transform s into t. To mention some significant examples, Levenstein [45] and Jaro-Winkler [72] are string-edit distances commonly used.

The count of deletions additions, substitutions or transpositions may be computed in different forms in each technique, for instance giving different weight to deletion and transposition operations, or even giving different weight to the different characters used in those operations. Another possible difference, tackled by Jaro-Winkler comparator, is giving more importance to coincidences in the first part of the string.

There is a subtype of character-based distances that tackle the problem from a different point of view: alignment functions. Some significant examples would be Smith-Waterman [63] and Needleman-Wunsch [51], both of them designed to compare biological sequences, such as amino acid chains. Those algorithms try to find the best possible alignment of both strings to find a result, i.d., they turn strings s and t into $s' = s'_0, s'_1, \ldots, s'_k$ and $t' = t'_0, t'_1, \ldots, t'_k$ where there are as much cases as possible in which $s'_i = t'_i$. To transform s into s' and t into t' *null characters* or *gaps*, denoted as λ, can be added at any position to s and t, but they will not count as a coincidence in case of $s'_i = t'_i = \lambda$. Needleman-Wunsch looks for global alignments between s and t, while Smith-Waterman looks for the best possible local alignment.

Token-based. In contrast to character-based functions, in which the strings s and t to compare are treated as an ordered sequence of characters, token-based functions consider s and t as a set or multi-set (bag) of words (tokens), without notions of order [14]. Those algorithms return a result based on the presence/absence of common tokens in s and t.

A simple example of this kind of function is the Jaccard similarity [34]. Considering strings s and t as sets of words S and T, the Jaccard similarity Δ_{JAC} between s and t would be $\Delta_{JAC}(s,t) = |S \cap T|/|S \cup T|$.

Another example is the use of the widely extended TF-IDF or cosine similarity [16].

Considering strings s and t as sets of words S of length n and T of length m, then cosine similarity Δ_{TFIDF} is defined as follows[16]:

$$\Delta_{TFIDF}(S,T) = \sum_{w \in S \cap T} V(w,S) \cdot V(w,T) \tag{1}$$

where $TF_{w,S}$ is the appearance frequency of word w in set S, IDF_w is the inverse of the fraction of names in the corpus that contains w,

$$V'(w,S) = \log(TF_{w,S} + 1) \cdot \log(IDF_w) \tag{2}$$

and

$$V(w,S) = \frac{V'(w,S)}{\sqrt{\sum_{i=0}^{n} V'(w_i,S)^2}} \tag{3}$$

Presenting this last algorithm as MERA-compatible may look controversial in the sense that we need extra info to reach a result than the pure strings s and t. We need to know the frequency of occurrence of each token of s and t as well as the total number of records in the corpus. Obviously, all this information could be found in the graph, but with large volumes of data it would be computationally prohibitive to navigate the whole graph looking for it in each comparison between two strings. Instead of that, the natural solution would be to query an extra structure in which that information may be computationally cheaper to obtain, such as an index of words combined with a counter of records. However, that structure must offer the same data that would be obtained by navigating the graph, so it should be completely synchronized with it.

The way in which that structure is maintained and queried is out of the scope of this work, but MERA has been designed to support that kind of logic for the algorithms plugged-in. Actually, we have implemented a prototype that uses the blocking function described in section 3.2.3, which uses an index of q-grams.

Hybrid distances. Some token-based functions works by using character based functions to give a result in order to lower the impact of small differences. Being Δ_{JAC} the function of Jaccard similarity, Δ_{LEV} the function of of Levenstein similarity, $s =$ "Shakira Isabel", $t =$ "Isabel Shakira", $t' =$ "Shakira Izabel" and $t'' =$ "Izabel Shakira", we would have the next:

- $\Delta_{JAC}(s,t) = 1$ and $\Delta_{LEV}(s,t) \approx 0$
- $\Delta_{JAC}(s,t') = 0.5$ and $\Delta_{LEV}(s,t') \approx 1$
- $\Delta_{JAC}(s,t'') = 0.5$ and $\Delta_{LEV}(s,t'') \approx 0$

Being Jaccard and Levenhstein representative token-based and character-based functions, we can see through this example that there are some situations in which none of this kind of functions can effectively detect the similarity of two strings that could actually be easily identified as referring to the same entity by a human. Token-based algorithms are critically affected when a typo is introduced in a word when dealing with short strings as s or t. Character-based algorithms usually return an inadequate result when the string to compare have disordered tokens. When both issues happen at time, as in the case of the pair (s,t''), then none of the algorithms is able to detect the pair as a match.

Hybird functions combine both kinds of algorithms (token based and character based) to be also effective in cases as the comparison of the pair (s,t''). Significant approaches of hybrid functions are Monge-Elkan comparator [48] and Soft TF-IDF [14].

Custom heuristics. There may be functions that do not exactly fit into any of the above described types but they can fit and they should fit in MERA's workflow. Those functions may be general-purpose or strongly connected to a particular case. Let's say that we want to link records of two sources A and B in which artists in A are stored with the initial of their names and their complete surnames, while artists in B are stored with their complete names and surnames. It might be a good idea to implement an ad-hoc function $\Delta_?$ in which given two strings of several tokens $s = s_0, \ldots, s_k$ and $t = t_0, \ldots, t_k$, and being each token t_i defined as $t_i = t_{i,0}, \ldots, t_{i,n}$ where $t_{i,j}$ is the character of position j in the token t_i, if $|s_{0,1}| = 1$, $s_{0,1} = t_{0,1}$ and $s_i = t_i \forall i \in [1,k]$, then $\Delta_?(s,t) = 1$. With this definition, $\Delta_?$("A Perry", "Albert Perry") $= 1$.

Any function able to map two strings s and t to a real number r in $[0,1]$ should fit in MERA's work-flow.

3.2.3. Blocking

MERA allows to use several blocking functions at a time. Some blocking techniques have been cited in section 2. However, some of these techniques are not as easy to include in MERA's workflow as the ones presented in sections 3.2.1 and 3.2.2, since the may need an extra structure than G in order to work efficiently.

There are blocking functions that may not need en extra data structure, such as the strategy of agreeing in some property to be considered a matching candidate [52]. Some other strategies, such as q-gram indexing [4] would need an index of q-grams built before executing the matching process. This new structure would not add any information different or more complete than the one included in G. It just must present slices of data in G to be accessed through computationally cheaper mechanisms.

This extra structure T should be completely synchronized with G. MERA expects to receive an already built graph G as an input and it will access it in read-only mode. With that, T can be built at the same time that G is built, or it can be created from scratch once G is not expected to receive more modifications. The execution of MERA will not introduce inconsistency between G and T, whenever the provided blocking functions use T also in read-only mode. Nevertheless, the process of building and updating G is not trivial, and it may get more troublesome if extra structures may reflect every change. However, best ways to tackle this problems are out the scope of this article and will be discussed in future work.

Algorithm 2 MERA Blocking

Require: Query q
Require: Index I
1: $N \leftarrow$ extractUniqueQrams(q.mainContent)
2: $E_{idf} = [\lambda]$
3: $E_{ngr} = [\lambda]$
4: **for all** $n_i \in N$ **do**
5: $\quad idf_{n_i} \leftarrow I$.idf($n_i$)
6: $\quad$ **for all** $e_i \in I$.entitiesWithQgram(n_i) **do**
7: $\quad\quad E_{idf}[e_i] \leftarrow E_{idf}[e_i]$+meraTfIdf($e_i, n_i, idf_{n_i}, I$)
8: $\quad\quad E_{ngr}[e_i] \leftarrow E_{ngr}[e_i] + 1$
9: result $\leftarrow \{\lambda\}$
10: **for all** $e_i \in E_ngr$ **do**
11: $\quad$ **if** $E_{ngr}[e_i] = |N|$ **then**
12: $\quad\quad$ result $\leftarrow$ result $+e_i$
13: **return** result $\cup$ bestK(E_{idf})

Proposed blocking function. We have designed and implemented in a prototype a blocking function adapted to MERA's management of alternative forms of an entity(it will be discussed in section 4), which consist in an adaptation of q-gram

indexing [4] and TF-IDF [58] that works as shown in algorithm 2. However, the pseudo-code needs some clarifications:

- Input *Query* q: Query for which we are searching the most similar candidates.
- Input *Index* I. Structure in which information about q-grams is stored, synchronized with Graph G.
- Function *extractUniqueQgrams*. It expects a string s and it returns a set containing all the q-grams n_i in s.
- Property *mainContent*. Invoked over a Query q, it returns the main string of q.
- Structure $\{\lambda\}$: represents an empty set.
- Structure $[\lambda]$: represents an empty dictionary, in which pairs (key $\rightarrow$ value) are stored.
- Method *idf*: when invoked over an Index I, it expects to receive a q-gram n_i and returns it the IDF score in Graph G.
- Method *entitiesWithNgram*: when invoked over an Index I, it expects an q-gram n_i and it returns the ID of all those entities in G that contains n_i.
- Function *meraTfIdf*: it expects an entity ID e_i, a q-gram n_i, an IDF score idf_{n_i} and an Index I. It returns the $r = tf \cdot idf_{n_i}$, where tf is the amount of forms of e_i in which the q-gram n_i appears. All this info should be accessible querying I.
- Function *bestK*: it expects a dictionary of elements and it returns the keys of the k pairs whit greater value. The parameter k should be specified through configuration.
- *Iterating over dictionaries*: when accessing the elements of dictionaries in *for loops*, we expect to receive in each iteration the key of a pair.

The differences between the algorithm 2 and classical TF-IDF approaches based in q-grams are:

- We introduce in the algorithm the concept of alternative form of an entity. An entity may be considered as a set of forms, and the algorithm cares about how many forms contain a q-gram, instead of how many total times an ngram is contained in all the entity forms.
- We manage the relevance of the q-grams in an entity with an accumulated TF-IDF score but also the number of q.grams contained in an entity e. If a certain entity contains all the q-grams of q.mainContent, then that entity is automatically included in the resulting set of candidates, not mattering the TF-IDF score or even the fixed limit of candidates defined through configuration.

The results of this technique will be discussed in section 4.

3.3. Graph navigation

MERA is designed to match queries with graph nodes that are representing entities. Finding a result requires the use of string similarity measures and graph navigation through nodes' neighbourhood. MERA uses two types of graph navigations when resolving a query: looking for alternative forms and looking for related entities.

Looking for alternative forms. An entity (node) e may be associated with several identifying or pseudo-identifying forms in a graph. Let's consider a Graph G containing a node e_α that represents the artist "Lady Gaga". The artist is represented by a single node e_α, but e_α may be associated with several pieces of string information that can be considered as identifying forms. This pieces could include canonical or artistic name ("Lady Gaga"), civil name ("Stefani Joanne Angelina Germanotta"), or known alias ("Mother Mosnter", "Jo Calderone"). Even typical typos may be useful identifying forms. Let's suppose that there is a tendency to misspell the name "Stefani", using instead "Stefany" or "Stephanie". Depending on the chosen reconciliation algorithms this information may result more or less relevant (phonetic encoding approaches will probably ignore that kind of issues). However, it could be also useful to store that kind of typos. Actually, some public sources about artist [30] store and offer that kind of data.

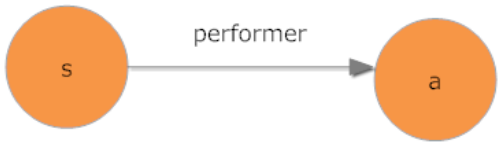
There could also be more complex types of alternative forms that imply graph navigation deeper than the search of several distinct properties linked to a node. An example of this may be the pertinence of an individual artist to a group. Due to metonymy issues, we could find songs wrongly associated with an individual when they should be associated with a group or vice versa. Then it could have sense to consider as an alternative form of an artist that belongs to a group the name of that group, or even all the alternative forms of that group.

The user of MERA should be able to specify the alternative forms to apply in each case, since they should be adapted to each specific context and data nature.

Looking for related entities. A MERA query is formed by a main content and a set of refinements or extra data. The output of MERA when receiving a query q is a set of nodes that represents the entities in G that have been detected to be more similar to the main content of q. However, the included refinements may refer to different entities related to the main one: For instance: artists or album of a song. These extra pieces of information are used for disambiguation purposes.

The provided extra pieces of information should be labelled, i.d., each one must be associated with a type. Let's say that MERA system tries to resolver a query q to find an artist with a name a, and the query also provides two names of songs s_1 and s_2 of a. Firstly, MERA system will select a subset S of all the nodes in G of type "artist" according to the specified blocking criteria. For each node n_i in S, MERA system will calculate a pre-score using a against the node's alternative forms. Then, it will explore the set of entities S'_i of type "song" linked to n_i, and it will use all the alternative forms of each entity in S'_i

Figure 2: Non MERA graph example



looking for coincidences with s_1 and s_2. If some coincidences above the user-specified thresholds for type "song" are found, then the score of q with n_i will increase.

There is no limit about the type or amount of extra data that may be provided in query q. Since we are working over rdf graphs, the types should be specified with a rdf property (or at least they should be mapped to a rdf property in a MERA implementation). Providing a type t in a query q that is not present in a Graph G should just return an empty set of results.

Graph form and sources management. MERA is designed to be able to manage different sources, storing the origin of each piece of information in a single Graph G. The strategy used to do that is working with a graph in which the nodes are not directly linked with their related contents (other entities of string properties), but auxiliary nodes are used as intermediaries. Those auxiliary nodes relate the subject and the object while they point to a third node that represents the origin of the information.

Let's say that we have a song s performed by an artist a. A natural way of representing this with an rdf graph would be the one shown in figure 2.

However, we also want to know *according to who* that song s is performed by artist a. Let's say that this information has been obtained from a dataset d. The way in which MERA stores that information is showed in figure 3. In rdf terms, a triple $t = (s, p, o)$ that has been obtained from a dataset d is transformed in three triples $t_0 = (s, p, n_{aux})$, $t_1 = (n_{aux}, p_{aux}, o)$ and $t_2 = (n_{aux}, p_{dat}, d)$, being n_{aux} a new auxiliary node, p_{aux} a special predicate "target" and p_{dat} the property "dataset".

As shown in figure 3, s is no longer pointing directly to a, but to aux node, being aux who points to a through the property "target". At the same time, aux points to d indicating from where the original triple has been obtained. In MERA's graph schema, nodes do not directly point to the rest of nodes or objects they should be naturally linked to, but to auxiliary nodes using the predicate they would have used if they had directly pointed to the target node.

Let's say that we want to add the info of a new dataset denoted as e to graph G and that according to e song s also has a as a performer. The resulting

Figure 3: MERA graph example

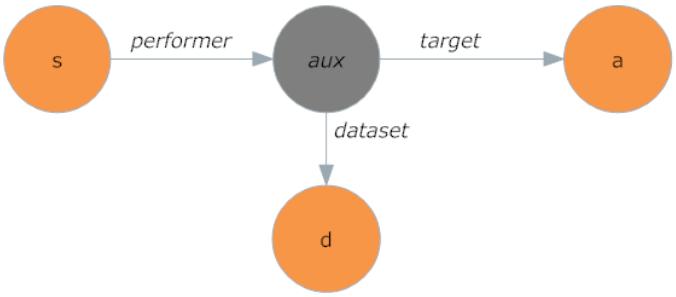
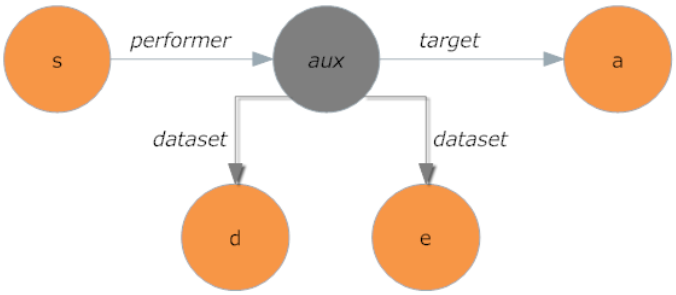


Figure 4: Auxiliary node with several datasets



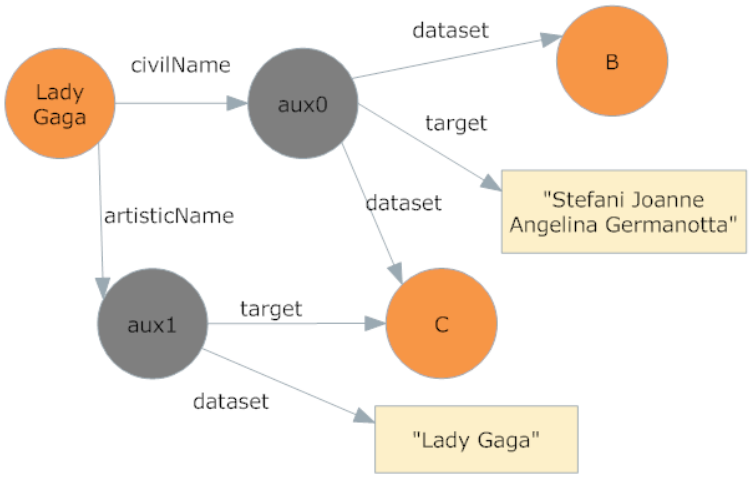
MERA graph would be the one shown in figure 4. The relation we want to represent is the same as in the previous case, i.e., the triple (s,"performer", a), but there is an already existing auxiliary node aux representing that link. So, we just add the triple (aux,"dataset", e) to indicate that dataset e also agrees about that link.

Information about sources can be used in two ways while computing or interpreting results. On the one hand, MERA allows the user to specify black or white lists of sources to be considered during the matching process. On the other hand, the user could also filter MERA results once it has been received according to his criteria.

A motivational example to filter data about other sources when receiving the results instead of ignoring it in the computation process (which would mean less comparison operations) would be the following. Let's say we want to merge two sources A and B and entities of A are put into queries while entities in B are dumped to a Graph G with MERA's schema. Set A contains an artist entity a_α identified as "Lady Gaga", and B contains an entity b_α identified as "Stefani Joanne Angelina Germanotta". We enrich graph G using a third source C that stores both forms as identifiers of the same person, letting us building a graph as the one shown in figure 5.

On the one hand, if we ignore the triples of C during the matching process, we will be in a situation in which we do not have enough information to match a_α

Figure 5: Artist described with two sources



and b_α, since there are not string similarity algorithms of general purpose that are able to recognize "Lady Gaga" and "Stefani Joanne Angelina Germanotta" as equal or similar. On the other hand, if we use all the graph nodes in the process, a_α will have a match with one of the forms of Lady Gaga's node. When receiving the results, we can check if the obtained node use some pieces of information of B, not mattering if those pieces have been used to do the match or not. If pieces of B are used, we have a pertinent result. If not, i.e., if the returned node only has pieces of information linked to C, the result can be ignored.

Creating graphs. The process of building graphs representing the information of one or several sources that was not originally presented in graph form is not a trivial task. However, the difficulty of this process may not be the same in different scenarios. Merging several sources in a graph may result troublesome since matching algorithms as the one presented in this work should be used. Creating a graph using a single source with internal unique identifiers may be easier since there is no ambiguity with which records represent which particular entities. However, in this work we are assuming the existence of an already built graph that satisfies MERA's schema, not mattering its origin or creation process. We plan to discuss how to properly create this structures in future work.

3.4. Configuration

In table 1 we have compiled and explained most of the configuration options of MERA's design.

Comparison configurations. As shown in table 1, most of MERA's configuration options are referred to comparison techniques to apply in different scenarios,

Table 1: Configuration Table

Concept	Range	Default
General defaults		
• **Per entity type**		
— (D1) Pre-processing algorithms	List of algorithms	None
— (D2) Comparison algorithms	List of algorithms	Exact match
— (D3) Threshold	[0,1]	0.65
— (D4) Relevance	[0,1]	0.70
• **Per order type**		
— (D5) Main type threshold	[0,1]	0.65
— (D6) Total threshold	$\mathbb{R}^+$	0.95
Type defaults		
— (D7) Pre-processing algorithms	List of algorithms	D1
— (D8) Comparison algorithms	List of algorithms	D2
— (D9) Threshold	[0,1]	D3
— (D10) Relevance	[0,1]	D4
Blocking		
— Top candidates blocking functions	$\mathbb{N}^+$	all
— Top MERA results per query	$\mathbb{N}^+$	10
— (D11) Blocking functions	List of blocking functions	None
Filtering sources		
— Type	{“White list”, “Black List”, “None”}	None
— Source list	Set of dataset URIs in G	None
Order configuration		
— Main type threshold	[0,1]	D5
— Total threshold	$\mathbb{R}^+$	D6
— Pre-processing algorithms	List of algorithms	D1
— Comparison algorithms	List of algorithms	D2
— Blocking functions	List of blocking functions	D11
• **Per refinement type**		
— Pre-processing algorithms	List of algorithms	D7
— Comparison algorithms	List of algorithms	D8
— Threshold	[0,1]	D9
— Relevance	[0,1]	D10

letting the user specifying algorithms, similarity thresholds, and relevance of types. There is a set of values in table's section *General defaults* which are used in case the user does not overwrite them for particular orders or types. We explain the meaning of these options in the following list:

- *Per entity type.* We are defining values to use when refining a query with extra data of a type t.
  - *Pre-processing algorithms.* Set of algorithms to use in pre-processing stage.
  - *Comparison algorithms.* Set of algorithms to use in comparison stage.
  - *Threshold.* When comparing two entities of type t, if none of the algorithms gives a result equal or greater than this value, the score will be ignored.
  - *Relevance.* When comparing two entities of type t in a refinement of a query, in case of finding a match high enough (score $\geq$ threshold), this match score will be added to the general query score after multiplying it by this factor of relevance.
- *Per order type.* This values are applied to the general score of a query or for operating with the main type of a query.
  - *Main type threshold.* Matches lower than this threshold with the main content of the query will be discarded.
  - *Total threshold.* If the result of the main type score plus the refinements score is lower than this value, the match will be discarded.
  - No default algorithms options are provided for order types. The default algorithms should be the ones assigned by default to the main type of the query.

A MERA implementation should follow a hierarchical configuration in which most of the matching aspects can be tuned but also all of them should offer a default value to use with the inheritance schema shown in table 1. Every query type can define its own values, both for the main type t of the query and for each type t_i of each refinement included.

Cutting number of results. MERA users may specify the number of candidate entities to use when receiving the results of a query or when filtering candidates for deeper comparisons. The default number of candidates when ending the blocking function is *all elements in G of type t* since the default blocking strategy does not include any blocking function. However, this parameter should be modified when including blocking functions.

Filtering sources. MERA's default planned behaviour is using all the data available in the graph, but the system should allow the user to discard a set of sources or to consider just a set of sources. In case of using black/white lists, the expected content of the list of sources should be URIs of datasets in graph G, or content that a MERA implementation could map to those URIs.

Specifying and adding algorithms. MERA's design is supposed to allow the user to develop and include in the system's workflow new algorithms in each process, so it should also allow the user to refer those new algorithms when configuring MERA. On the one hand, an implementation in which configurations are specified through code may solve this situation using callbacks. On the other hand, implementations in which configuration is supposed to be provided through files may use mechanisms such as naming conventions. Our implemented prototype, presented in section 4, is configured using JSON files, being the algorithms specified through a class name. That class name is mapped to the path of the file in which a class implementing certain comparison interface can be found.

Defining alternative forms. A configurable option not shown in table 1 is the specification of alternative forms per each type of node. It may be hard to propose default values for this since these properties may be strongly connected both to the data nature and the ontology used in graph G. For instance, possible alternative forms of nodes of type "artist" could be strings pointed by properties such as "civil name", "artistic name", "alias", "name variation"... the user should also be able to specify as an alternative form pieces of information accessed through deeper graph navigation. It would be feasible to consider as an alternative form of an artist the names of the groups from which he is member. Or even all the alternative forms of those groups. That is, "member of group" property and all those properties that has been configured to be alternative forms of nodes of type "group".

Default specified values. The default numeric values shown in table 1 are just a functional configuration of MERA that frees the user from having to configure every MERA options. However, since we assume that is hard to find standard scenarios, we would encourage the users of a MERA implementation to define their own default values according to their objectives (precision and recall) and the nature of the target databases.

4. Evaluation

We have implemented a prototype using the programming language python that covers most of MERA's specifications, including:

- Graph navigation exploring alternative forms for each entity.
- Graph navigation exploring related entities.
- Configuration of relevancies when applying refinements to a query.
- Configuration of minimum acceptable values for each type when scoring a result.
- Blocking function described in section 3.2.3.
- Usage of MERA rdf graph schema, described in section 3.3.

- Set of comparison algorithms of general-purpose.
- Set of text standardization functions.

We have used our prototype for testing MERA's proposals. Our experiment has consisted in the reconciliation of a random slice of two different musical sources A and B against a third source C. We have put the information of C in a graph G and we have used information of A and B to built queries. We have manually checked what should be the result for each query and we have measured the correctness of the the result returned by our prototype under different conditions. Source A is supposed to contain data of high quality (complete and with none or few misspellings) while source B is supposed to contain noisy data (user-entered and incomplete). However, every query thrown against G will be formed by a song name and one or more names of associated artists/writers.

We have thrown the same queries several times including/excluding some of MERA's features in order to check how this affect the results. We have maintained the same selection of algorithms during all the experiments, since we did not pretend to measure how the selected algorithms affected the result, but how MERA's ideas (two kinds of graph navigation and an adapted blocking function) works.

When our prototype receives a query q, it returns a ranked list of nodes representing entities that has been detected to be more similar to q. We have measured the results by annotating the rank of the expected node.

4.1. Experiments

We have executed our experiments under the following software/hardware conditions:

- Virtual Machine using Windows Azure cloud computing services.
- 64-bit Operating System: Windows Server 2012 R2.
- AMD Opteron (tm) Processor 4171 HE 2.10 GHz.
- 14 GB RAM.
- Python 2.7.3 64-bit.

4.1.1. Selection of data

Graph. We have used the source discogs [30] to create a graph G containing a total of 500.000 songs as well as their associated artists and writers. The information has been extracted from the dump of discogs releases published in 2015-01-01. That dump includes a set of musical releases containing one or more tracks and associated artist, with a total of 45.458.287 detected tracks among all releases.

The scalability of our prototype is not enough to manage a graph with the data of the the entire discogs dump since it completely works in main memory, so we have reduced that quantity to 500.000, a number that we can handle but

that is still a representative and high enough slice. Those 500.000 songs include 205 that has been manually detected to be the adequate answer to the queries used in the experiment; the rest of songs has been randomly selected using the python function *random.randInt()* to generate a set S of 499.795 random numbers between 1 and 45.458.287 (total number of discogs tracks). The songs with a relative position in the dump file contained in S where included in the graph.

We also included in G different artists and writers linked to each song. Discogs associates people to tracks/releases using different roles. We have mapped some of the original discogs roles to our own roles to include these associated people in our graph. Discogs roles "Artist", "Featuring", and "Vocals" have been mapped to "artist" , and "Written-By" has been mapped to"writer". The rest of discogs roles have been ignored.

All releases and artists have their own unique discogs ID, so there had been no need to use reconciliation techniques to avoid including repeated entities in the graph. The obtained graph, used in all the experiments, contains a total of 500.000 songs and 624.441 people.

Source of high quality. We have found that the open music encyclopedia MusicBrainz fits in our requirements of offering a great amount of data of high quality [65]. We have randomly selected a set of works from this source and we have manually checked that they point to a work that also exists in discogs data dump. We have selected the firsts 100 coincidences and we have used them to build queries. The average of associated artist/writers per song is 2.62

Noisy source. In 2006, AOL Inc. released a file containing twenty million search keywords for over 650.000 users [3] over a three-month period. We have considered that musical-related searches found among this material could be a representative example of a noisy source, since all the inaccuracies have been introduced by random real users when doing real searches. Our strategy to find musical content among the rest of searches has followed the next steps:

- We explored all the items selecting those that contained the tokens "feat" and "featuring". We splitted those items in two parts using "feat" as separator, and we added each one to a set of noisy candidate keys N
- We cleaned the keys N by erasing words or sequences that are meaningless for our purposes, such as "to download", "ringtone", "free music", "song of", "video", etc. In case of finding " - " in the key, we splitted it again and repeated the cleaning process with each part. All the results were added to a set of clean candidate keys C with 1203 elements.
- We revised manually each key in C in the context of its apparition in the log of AOL to check if they really were artists/songs or not. In addition, if we detected that some of the keys contained more than one artist/song at a time, we splitted it in parts and considered it separately. At the end of this process 922 sequences (not unique) were detected to contain songs or artists. We stored it in a set of found keys F.

- We put all the elements of F in a set of definitive keys D. Then, for every key k_i in F with a length greater than 3, we generated all the possible strings $S_i = \{s_0, s_1 \dots s_k\}$ in a Levenshtein distance of one with k. For each $s_j \in S_i$, if s_j was found more than 5 times in the log of AOL, it was added to D. With this, we collected a group of detected musical entities that were queried by users as well as character variations of all of them that has a certain presence in the log of AOL. We remove from D meaningless sequences as the ones used in previous steps.

- We explored again the entire log looking for searches containing at least a key of D. We elaborated three different lists with the found searches, regarding if they contained a single key, between two and three or more than four. 327.904 searches were detected.

- We messed those lists randomly using the function *random.shuffle()* of python . Then, we revised manually the lists to extract the first 35 results of each one that we could identify as searches for songs that included, at least, one artist name. We erased all the meaningless words or sequences of previous steps from the results.

In every cases, we respected the original appearance of each entity in the log of AOL. That is, we identified different parts in each search and we remove meaningless words, but no other changes were applied to the original content.

With this, we obtained a list of 105 lines containing queries with noise introduced by the users. The average of associated artist per song is 1.44.

4.1.2. Execution of experiments

We have executed a set of experiments designed to check the impact of MERA's techniques over scenarios with different level of data quality and we have measured the quality of the obtained results using different subsets of MERA's features. For all experiments we have used the same configuration if proceed, i.e., if the corresponding MERA functions for this configuration are active during the test.

- Blocking function the best 60 candidates based in accumulated TF-IDF score of q-grams. In case of being active MERA blocking function as described in algorithm 2, the candidates may exceed that number if there are enough individuals containing all the q-grams of the song provided in the query.

- Top results per query: 15.

- Minimum score for a song to be accepted: 0.50.

- Minimum score for an artist to be accepted: 0.65.

- Artist relevance when refining the result of a song: 0.80.

- Reconciliation algorithms used in all cases: Levenshtein similarity.

- Pre-processing functions included: replacement of all non-ASCII characters by ASCII equivalents with the function unicodedata.normalize("NFKD", targetString) of python, lower-casing, deletion of punctuation marks, and deletion of redundant white spaces.

This configuration has been thought to allow relatively bad results to be ranked (in a low position) as possible matches, since we are interested in finding how the target result ranks in every case. An execution of MERA looking for safe matches should probably define higher values of candidates acceptance. The reason for which we have chosen just one reconciliation algorithm is to check how MERA's novel ideas impact over results, not the effectiveness of the particular used algorithms in each case.

The alternative forms that we have considered in each case when building the graph G has been the next:

- Artists: Canonical name according to discogs, civil name, alias and usual name variations. If we were dealing with a group, also the canonical name of its integrating artist. If we were dealing with a person, the canonical name of the groups he/she belongs to. All this information is offered by discogs.

- Songs: Canonical name according to discogs. In case that this name had text between brackets (ex: "Summer love (radio remix)"), the same name without brackets has been considered as an alternative form.

The following combinations has been tested:

- *Experiment A*: reconciliation of songs without using alternative forms nor related artist. Not using MERA blocking function. Results are shown in the chart 6a.

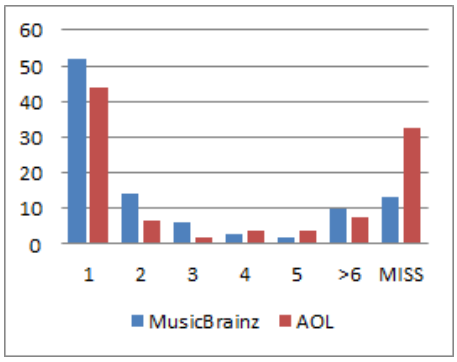
- *Experiment B*: Reconciliation using alternative forms of songs, but without using information of related artists. Not using MERA blocking function. Results are shown in the chart 6b.

- *Experiment C*: reconciliation of songs using related artist, but without using alternative forms of each entity. Not using MERA blocking function. Results are shown in the chart 6c.

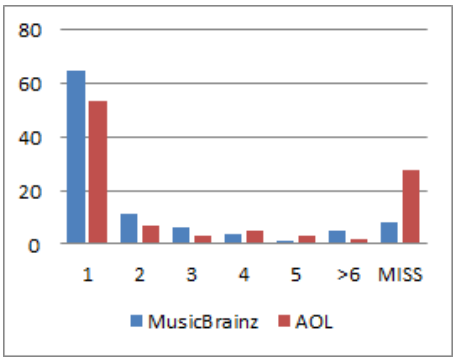
- *Experiment D*: reconciliation using both kinds of graph navigation, but without using MERA blocking function. Results are shown in the chart 6d.

- *Experiment E*: Reconciliation using both kinds of graph navigation and also MERA blocking function. Results are shown in the chart 6e.

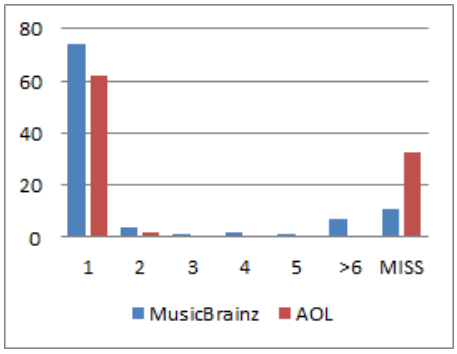
We also have included a comparison of the difference between using a blocking strategy of q-gram indexing with TF-IDF and using MERA blocking function for each source. Results for AOL are shown in the chart 7a and results for MusicBrainz in the chart 7b.



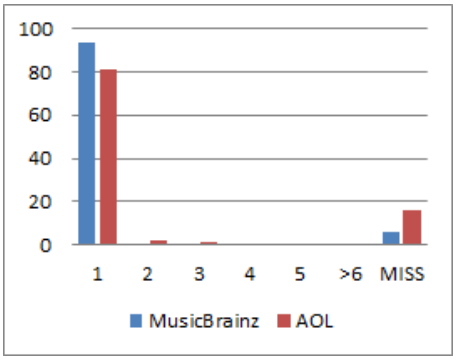
(a) No graph navigation



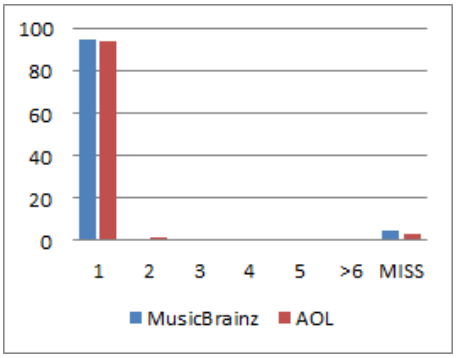
(b) Alternative forms navigation



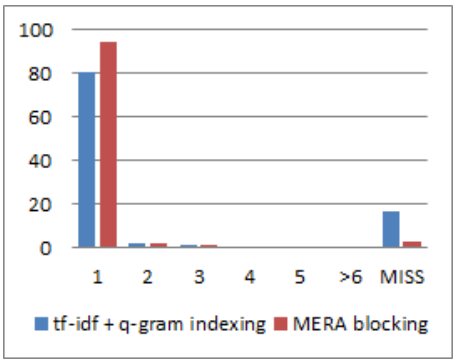
(c) Related entities navigation



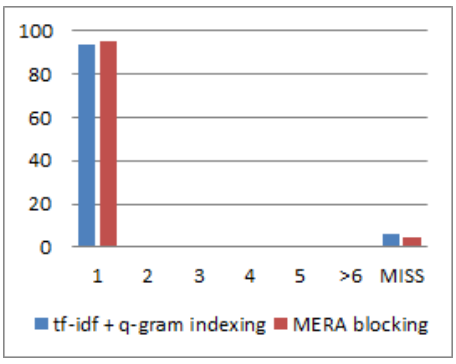
(d) Both navigation ways



(e) Both navigation ways



(a) Blocking strategy in AOL



(b) Blocking strategy in MusicBrainz

4.2. Discussion

4.2.1. Inclusion of MERA features

Experiment A. No graph navigation. As shown in the chart 6a, 52% of MB[3] queries has been detected in first place, 35% has been detected in intermediate positions and 13% has not been detected. In the case of AOL queries, 43.81% in first position, 23.81% in intermediate positions and 32.38% missed.

We have analysed the particular reasons for which the target result has not been detected by the prototype or has been detected in low ranks, and we found that those reasons are nearly the same for both sources:

- The blocking function did not detect as candidate the target result. This happens mostly when dealing with songs with a short name, with really common words in G or a combination of these two factors. Examples of this would be "'So what" of artist "Ciara" or "Bestfriend" of artist "50cent".

- Ambiguity: Song names are not unique in some cases. If we do not explore related artist in order to decide the best result there is not a reliable criteria to decide which result should appear in first place. An example of this would be the song "SOS" of "Rihanna", since many other songs with the same name has been stored in G.

- Lack of alternative forms of song. The titles of AOL and MB may present a great difference with their corresponding title in discogs according to Levenshtein distance, despite of the fact that they point to the same reality. This is mainly because discogs' content usually contains song titles with information between brackets, such as "Summer love (radio edition)" or "Summer love (remix DJ)". This information may be useful for disambiguation purposes, but it could also represent noise when trying to

[3] MusicBrainz

conciliate it with sources not so precise with song versions, as the queries built using MB/AOL data.

Experiment B. Graph navigation just looking for song alternative forms. As shown in the chart 6b, there is an improvement in the results of both sources compared with experiment A. The rate of firsts results of MB grows from 52% to 65%, the rate of intermediate results decreases from 14% to 11% and the rate of missed results decreases from 13% to 8%. In the case of AOL the rate of first results grows from 43.81% to 53.33%, intermediate results descend from 23.81% to 19.05% and missed results decreased from 32.38% to 27.62%.

Analyzing particular cases, we found that the reasons for this improvement has been nearly the same for both sources. Using graph navigation to find alternative forms of songs let us to relate queries such as "Touch it" with entries in discogs such as "Touch it (remix radio edition)", since the text between brackets has been erased to introduce in the graph the alternative form "Touch it". However, this also has the effect of the apparition of more false positives. Despite of the fact that the general results has improved, there are some queries that has ranked worse than in Experiment A. This is because some songs that had not been detected as matches in A(and should not have been detected) are already detected in conditions of experiment B.

Experiment C. Graph navigation looking for alternative forms. As shown in the chart 6c, when we use graph navigation to look for artists associated to a song, the rates of both sources improve compared to experiment A. In the case of MB, the rate of firsts found results grows from 52% to 74%, and the rate of intermediate results decrease from 35% to 15%. In the case of AOL, the rate of firsts results grows from 43.81% to 61.9% and the rate of intermediate results decrease from 23.81% to 5.7%.

For both sources, results of experiment C have improved compared to the results of experiment B. That is, graph navigation looking for related entities seems to have more positive impact over the results than graph navigation looking for alternative forms. In the case of AOL, firsts results rate grows to 53.33% in B but to 61.9% in C. For MB, first results grows to 65% in B but to 74% in C. However, there is not a significant improvement, or even any improvement at all in the case of AOL, regarding to missed results if we compare experiments A and C. The rate of missed results in MB does decrease from 13% to 11%.

The reasons found for the improvements in first results rate is that, in most of the cases in which there were ambiguity deciding which song should appear first, now we have the artist name as a disambiguation mechanism. Some results are still missed because not using alternative forms do not allow the prototype to recognize songs with extra information between brackets, since that extra characters make the difference applying Levenshtein distance. The fact that the rate of missed results remained the same compared to 6a is due to entities not passing the cut of the blocking function as it is configured. Since no alternative forms has been added to the graph, the presence of q-grmas in each song remains

exactly the same than in case 6a, so the nodes obtained in blocking stage are also the same.

Experiment D. Both graph navigation types. As shown in the chart 6d, when using both graph navigation types, MB firsts result rate grows up to 94% and missed results rate decrease to 6%. No intermediate results at all appeared. In the case of AOL, firsts rate grows up to 80.95% and missed results descend to 16.19%, with a rate of intermediate results of 2.85%.

For both sources, the conditions of experiment D drives into a significant improvement compared to previous tests. However, exploring why there are still results missed or under ranked, we find that there are different reasons for each source.

- MB: there are 6 results not ranked first. One of them did not pass the cut of the blocking function because the song has a short name formed by common q-grams in Spanish: "El amor". The other five are all pieces of classical music in which it looks like Levenshtein distance is not enough to localize equivalences. An illustrative example of this is the query "Sonata for Piano no. 7 in C major, K. 284b/309: III. Rondeau. Allegretto grazioso". Its equivalent in discogs express the same reality but with different naming conventions. Artist information has not been useful in these cases since the classical writers provided by MusicBrainz for those pieces (Mozart, Beethoven,...) are widely repeated. It looks like different algorithms should be applied for these kind of works.

- AOL: The under ranked results (position 2 to 15) appear mainly because of the detection of versions of songs before the one we are looking for. The reason for the missed results is, in all cases, that the target results did not pass the cut of the blocking function. Some results are still missed because the song name contained hard misspellings. An illustrative example would be the string "ghostrider" trying to express the song "Ghost Writer".

Experiment E. Including MERA blocking. When we use both kind of graph navigations and also the blocking function described in 3.2.3, then all the target nodes pass the blocking cut. As shown in the chart 6e, results compared with experiment D improve for both sources. However, as shown in the charts 7a and 7b, the improvement is more significant in AOL. In the case of MB, the only work that had not pass the blocking cut in D appears now first-ranked. The classical pieces score the same as in D. In the case of AOL, firsts result rate grows from 80.95% to 94.29%, and missed results rate decrease from 16.9% to 2.86%. For both sources, firsts result rate reach quotes higher than 94%.

Nevertheless, this improvement has a performance cost. We repeated 50 times the experiments E and D and our measurements indicate that the performance is 14.76% worse in E. We think the performance of this function may be strongly connected to data nature. We have measured how MERA blocking function exceeds the preconfigured number of candidates to be accepted in blocking stage in experiment E. That number had been set to 60, and we checked

that the average number of candidates returned per entity has been 68.12. Also, 94.62% of the queries had exactly 60 results. However, we have measured that in some cases the function had returned a quantity of candidates that exceed by large 60. This had happened with queries formed by short names with common q-grams in G. While executing E, the query that produced a higher number of valid candidates in blocking stage has been "So what" of the artist "Ciara", with a total of 326 detected candidates.

4.2.2. Data quality effect

As shown in charts 6a, 6c, 6b and 6d, if we do not use all the MERA features implemented in our prototype, the results of the clean source (MB) improves the results of the noisy source (AOL). However, when using all of them at a time, as shown in the chart 6e, both kind of sources present very similar result, even having AOL a lower rate of missed songs (2.86% vs 5% of MB). Nevertheless, this measurements may be conditioned by the size and nature of the samples.

5. Conclusions and future work

We have provided MERA, the architecture of a highly configurable system for matching several data sources containing musical metadata. Our design introduces some new concepts/ideas over existing general-purpose or music-specialized systems:

- Graph-based approach in which every piece of information maintains a list of source(s) that agrees on it, allowing the user to use this information for filtering contents.
- Configurable graph navigation to look for alternative identifying forms of each node to use in the matching process.

We also have proposed a blocking technique based on q-gram indexing and TF-IDF adapted to MERA's graph schema. We implemented a prototype including these features and we found that it was effective in more than 94% of the cases under the conditions of our experiments, which included sources with different level of data quality.

Many work still needs to be done so as to produce a final reusable system that catches all the presented ideas on this work. Nevertheless, we are optimistic with the collected results of our prototype, and we are currently working on the implementation of that system in the form of a framework for the programming language python. We are following several work lines:

- Work describing forms of building the graph that the algorithm of MERA uses.
- Deep testing for comparing a MERA implementation against existing general-purpose and music-specialized systems.
- Testing MERA's ideas with data not linked to the music world.

- Incorporation of supervised matching techniques through training data.
- Incorporation of parallel processing mechanisms.
- Scalability improvement for graph storage and management.
- Efficiency improvement for graph access.
- Study of existing ontologies or creation of a new one to produce clean and reusable graphs.

6. References

[1] I. Amón and C. Jiménez. Funciones de similitud sobre cadenas de texto: una comparación basada en la naturaleza de los datos. In *CONF-IRM Proceedings*, volume 58, 2010.

[2] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 586–597. VLDB Endowment, 2002.

[3] M. Arrington. Aol proudly releases massive amounts of private data. *TechCrunch: http://www. techcrunch. com/2006/08/06/aol-proudly-releasesmassive-amounts-of-user-search-data*, 2006.

[4] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD*, volume 3, pages 25–27. Citeseer, 2003.

[5] T. R. Belin and D. B. Rubin. A method for calibrating false-match rates in record linkage. *Journal of the American Statistical Association*, 90(430):694–707, 1995.

[6] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal - The International Journal on Very Large Data Bases*, 18(1):255–276, 2009.

[7] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):5, 2007.

[8] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[9] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.

[10] M. Bilgic, L. Licamele, L. Getoor, and B. Shneiderman. D-dupe: An interactive tool for entity resolution in social networks. In *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, pages 43–50. IEEE, 2006.

[11] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 313–324. ACM, 2003.

[12] P. Christen. Febrl-: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1065–1068. ACM, 2008.

[13] P. Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Springer, 2012.

[14] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. *Kdd workshop on data cleaning and object consolidation*, 3:73–78, 2003.

[15] W. W. Cohen. A web-based information system that reasons with structured collections of text. In *Proceedings of the second international conference on Autonomous agents*, pages 400–407. ACM, 1998.

[16] W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems (TOIS)*, 18:288–321, 2000.

[17] A. Culotta and A. McCallum. Joint deduplication of multiple record types in relational data. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 257–258. ACM, 2005.

[18] H.-H. Do and E. Rahm. Coma: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 610–621. VLDB Endowment, 2002.

[19] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 85–96. ACM, 2005.

[20] U. Draisbach and F. Naumann. Dude: The duplicate detection toolkit. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2010.

[21] T. E. Dunning and B. D. Kindig. Determining a known character string equivalent to a query string, June 9 2009. US Patent 7,546,316.

[22] T. E. Dunning, B. D. Kindig, S. C. Joshlin, and C. P. Archibald. Associating and linking compact disc metadata, July 19 2011. US Patent 7,984,062.

[23] A. Elmagarmid, I. F. Ilyas, M. Ouzzani, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. Nadeef/er: Generic and interactive entity resolution. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1071–1074. ACM, 2014.

[24] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[25] G. A. Fink. *Markov models for pattern recognition: from theory to applications*. Springer Science & Business Media, 2014.

[26] W. R. Gilks. *Markov chain monte carlo*. Wiley Online Library, 2005.

[27] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, D. Srivastava, et al. Approximate string joins in a database (almost) for free. In *VLDB*, volume 1, pages 491–500, 2001.

[28] S. Guha, N. Koudas, A. Marathe, and D. Srivastava. Merging the results of approximate match operations. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 636–647. VLDB Endowment, 2004.

[29] P. A. Hall and G. R. Dowling. Approximate string matching. *ACM computing surveys (CSUR)*, 12(4):381–402, 1980.

[30] J. Hartnett. Discogs. com. *The Charleston Advisor*, 16(4):26–33, 2015.

[31] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *ACM SIGMOD Record*, volume 24, pages 127–138. ACM, 1995.

[32] T. N. Herzog, F. J. Scheuren, and W. E. Winkler. *Data quality and record linkage techniques*. Springer Science & Business Media, 2007.

[33] D. Holmes and M. C. McCabe. Improving precision and recall for soundex retrieval. In *Information Technology: Coding and Computing, 2002. Proceedings. International Conference on*, pages 22–26. IEEE, 2002.

[34] P. Jaccard. *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz, 1901.

[35] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[36] M. A. Jaro. Probabilistic linkage of large public health data files. *Statistics in medicine*, 14(5-7):491–498, 1995.

[37] P. Jokinen, J. Tarhio, and E. Ukkonen. A comparison of approximate string matching algorithms. *Software: Practice and Experience*, 26(12):1439–1458, 1996.

[38] P. Jurczyk, J. J. Lu, L. Xiong, J. D. Cragan, and A. Correa. Fril: A tool for comparative record linkage. In *AMIA annual symposium proceedings*, volume 2008, page 440. American Medical Informatics Association, 2008.

[39] D. V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems (TODS)*, 31(2):716–767, 2006.

[40] H. Kang, L. Getoor, B. Shneiderman, M. Bilgic, and L. Licamele. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(5):999–1014, 2008.

[41] D. E. Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[42] H.-C. Kum, A. Krishnamurthy, A. Machanavajjhala, M. K. Reiter, and S. Ahalt. Privacy preserving interactive record linkage (ppirl). *Journal of the American Medical Informatics Association*, 21(2):212–220, 2014.

[43] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[44] M. Larsen. Multiple imputation analysis of records linked using mixture models. In *Statistical Society of Canada Proceedings of the Survey Methods Section*, pages 65–71, 1999.

[45] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[46] B. Lynch and W. Arends. Selection of a surname encoding procedure for the statistical reporting service record linkage system. *Washington, DC: United States Department of Agriculture*, 1977.

[47] A. McCallum and B. Wellner. Object consolidation by graph partitioning with a conditionally-trained distance metric. In *KDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*. Citeseer, 2003.

[48] A. E. Monge, C. Elkan, et al. The field matching problem: Algorithms and applications. In *KDD*, pages 267–270, 1996.

[49] A. Mount. Allmusic. http://www. allmusic. com/. *Journal of the Society for American Music*, 7(03):359–361, 2013.

[50] G. Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[51] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970.

[52] H. B. Newcombe. *Handbook of record linkage: methods for health and statistical studies, administration, and business*. Oxford University Press, Inc., 1988.

[53] H. B. Newcombe and J. M. Kennedy. Record linkage: making maximum use of the discriminating power of identifying information. *Communications of the ACM*, 5(11):563–566, 1962.

[54] T. Peng, L. Li, and J. Kennedy. A comparison of techniques for name matching. *Journal on Computing (JoC)*, 2(1), 2014.

[55] L. Philips. The double metaphone search algorithm. *C/C++ users journal*, 18(6):38–43, 2000.

[56] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[57] V. Rastogi, N. Dalvi, and M. Garofalakis. Large-scale collective entity matching. *Proceedings of the VLDB Endowment*, 4(4):208–218, 2011.

[58] G. Salton and M. J. McGill. Introduction to modern information retrieval. 1983.

[59] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278. ACM, 2002.

[60] M. Sariyar and A. Borg. The recordlinkage package: Detecting errors in data. *The R Journal*, 2(2):61–67, 2010.

[61] R. Schnell, T. Bachteler, and S. Bender. A toolbox for record linkage. *Austrian Journal of Statistics*, 33(1-2):125–133, 2004.

[62] R. Schnell, T. Bachteler, and J. Reiher. Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, 9(1):41, 2009.

[63] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[64] A. R. Stutzbach. Musicbrainz (review). *Notes*, 68(1):147–151, 2011.

[65] A. Swartz. Musicbrainz: A semantic web service. *Intelligent Systems, IEEE*, 17(1):76–77, 2002.

[66] J. R. Talburt. *Entity resolution and information quality*. Elsevier, 2011.

[67] S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems*, 26(8):607–633, 2001.

[68] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 350–359. ACM, 2002.

[69] D. Vatsalan, P. Christen, and V. S. Verykios. A taxonomy of privacy-preserving record linkage techniques. *Information Systems*, 38(6):946–969, 2013.

[70] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk-a link discovery framework for the web of data. *LDOW*, 538, 2009.

[71] W. Winkler. Issues with linking files and performing analyses on the merged files. *Proceedings of the Sections on Government Statistics and Social Statistics, American Statistical Association*, pages 262–265, 1999.

[72] W. E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. 1990.

[73] W. E. Winkler. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*. Citeseer, 1999.

[74] W. E. Winkler. Methods for record linkage and bayesian networks. Technical report, Technical report, Statistical Research Division, US Census Bureau, Washington, DC, 2002.

[75] W. E. Winkler. Overview of record linkage and current research directions. In *Bureau of the Census*. Citeseer, 2006.

[76] W. E. Yancey. Bigmatch: A program for extracting probable matches from a large file for record linkage. *Computing*, 1:1–8, 2002.

LaTeX Source Files

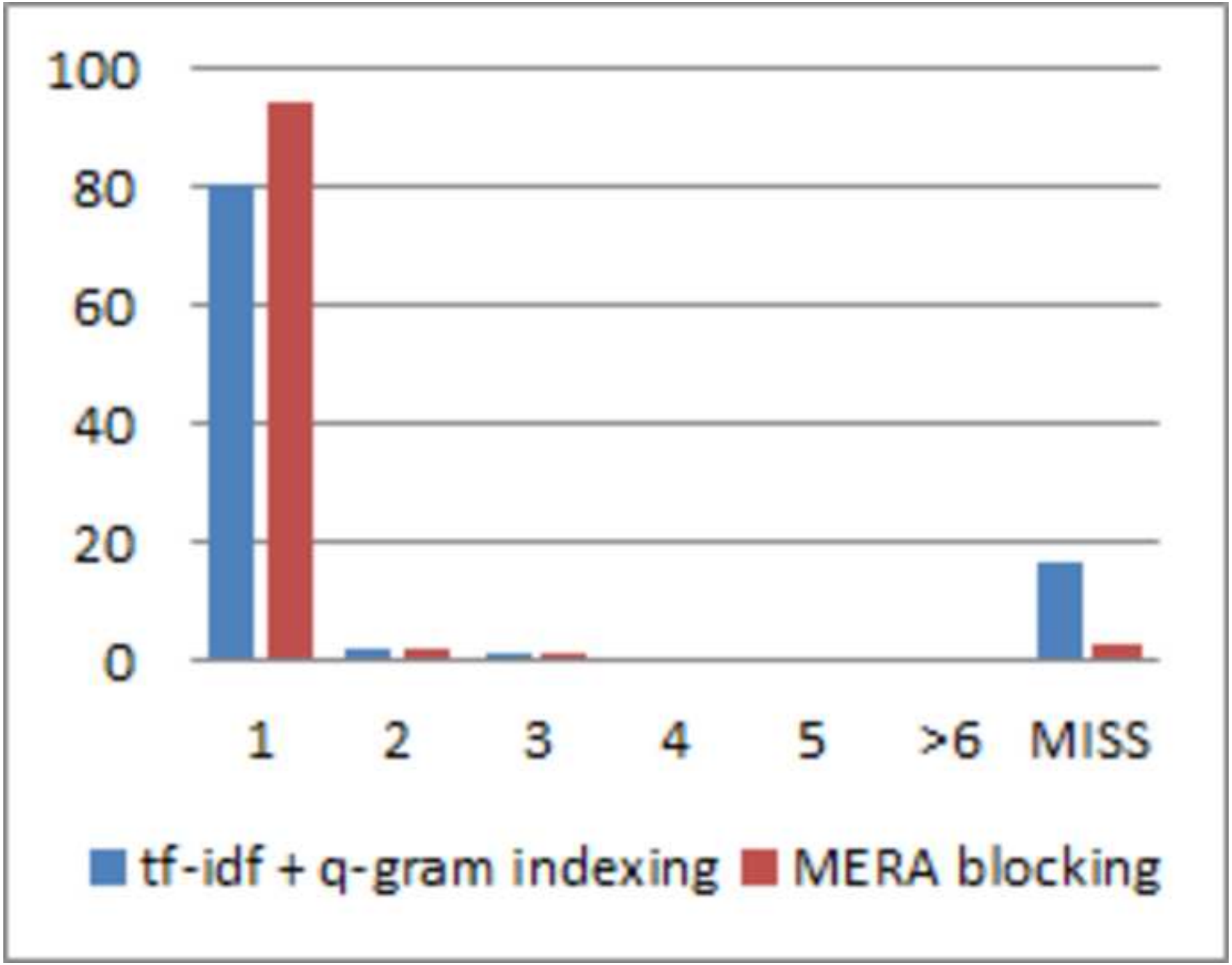
Click here to download LaTeX Source Files: LATEX source files.zip

***Highlights (for review)**

- We design an architecture for record linkage of entities related to the music world.
- The architecture let the user adapt the matching process to different conditions.
- We use rdf graphs with a novel schema to represent the information.
- We navigate graphs with our adapted schema during the reconciliation process.
- We experiment with a prototype that implements most of the described specifications.

Figure

Click here to download high resolution image



Figure

Click here to download high resolution image

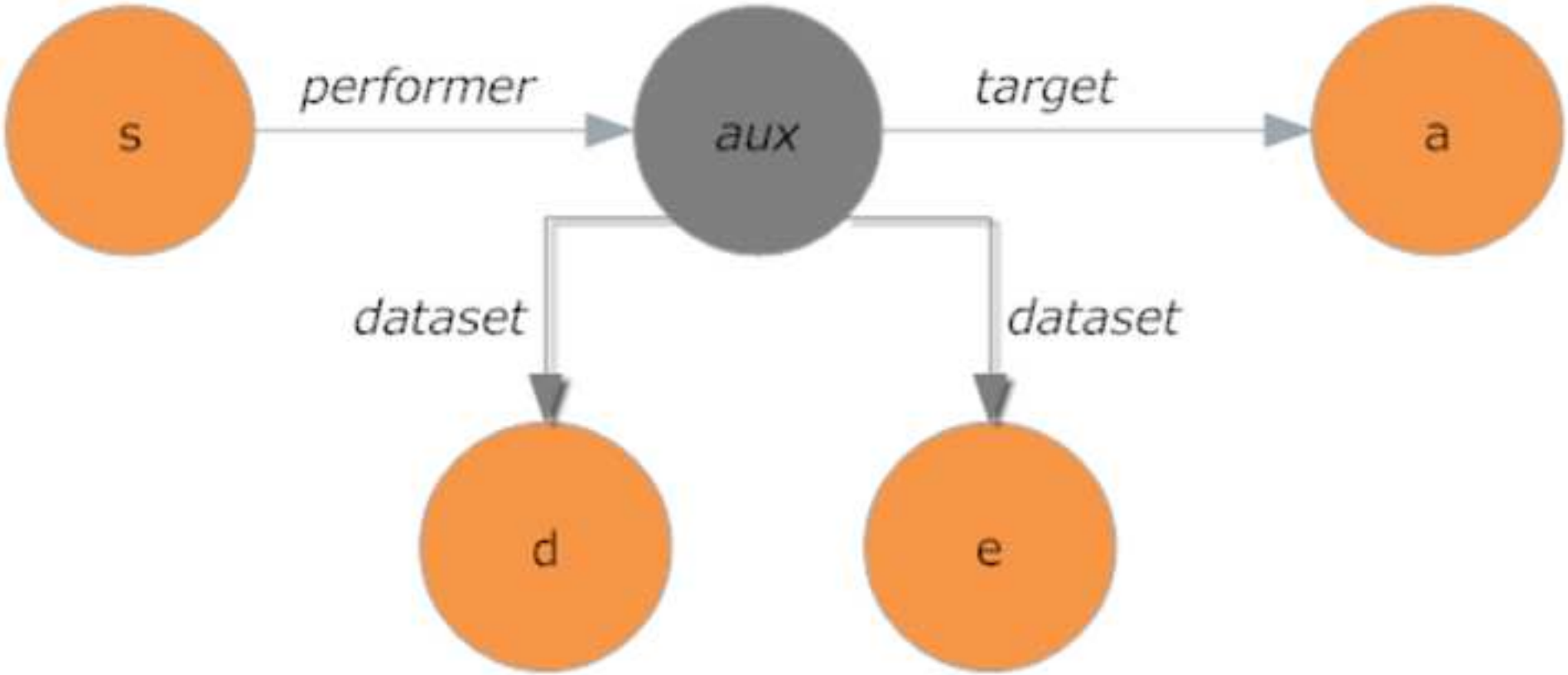
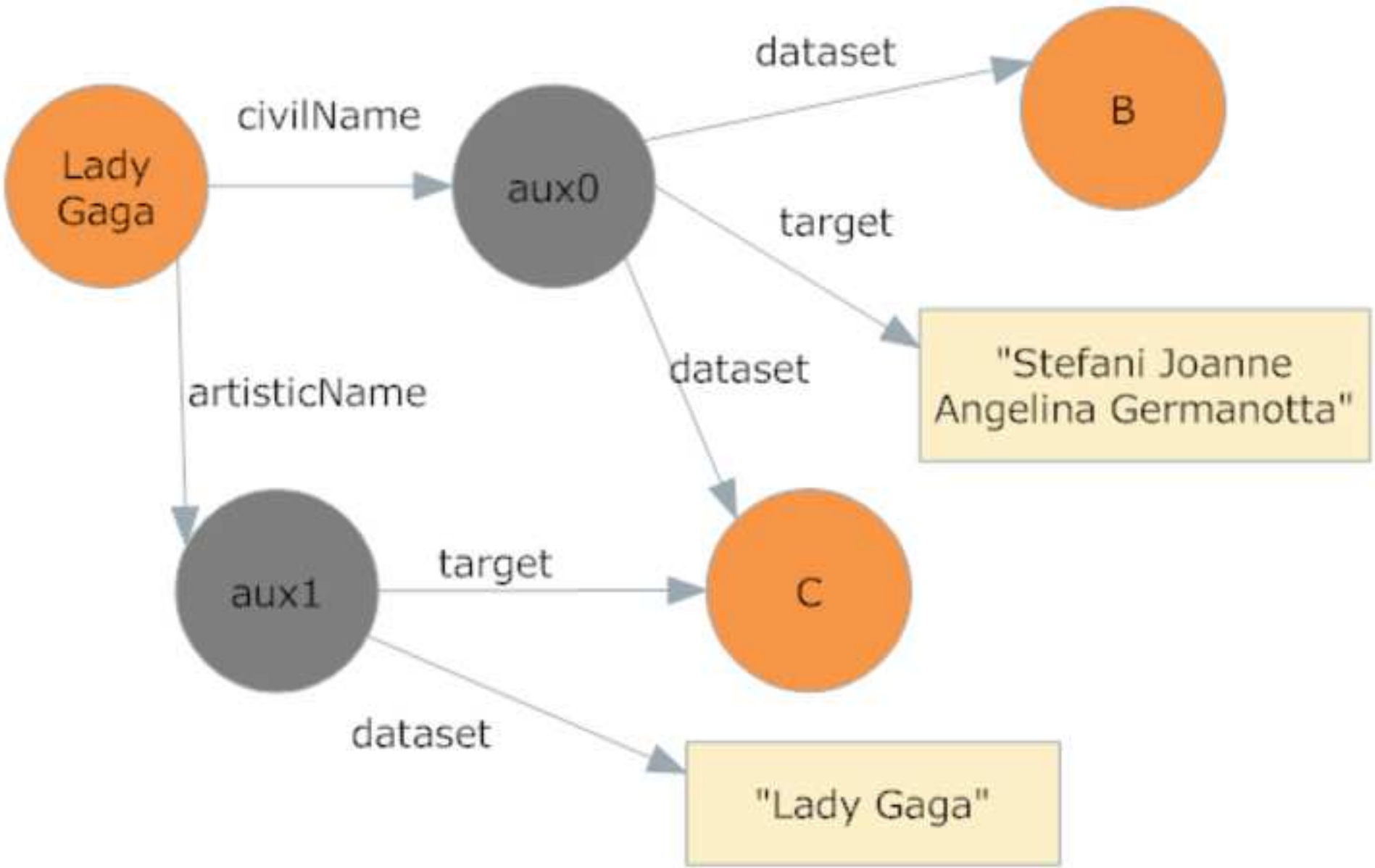
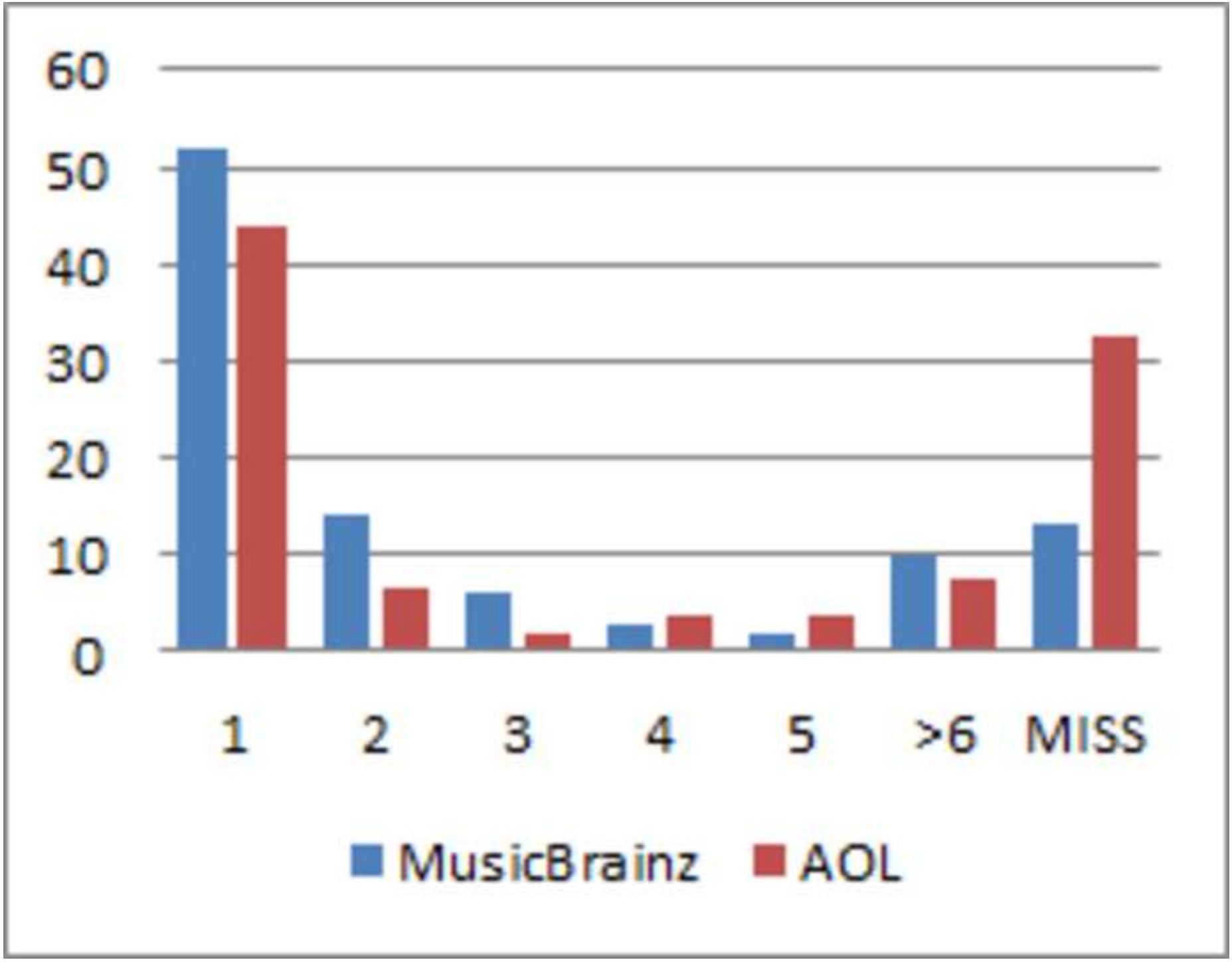


Figure
Click here to download high resolution image



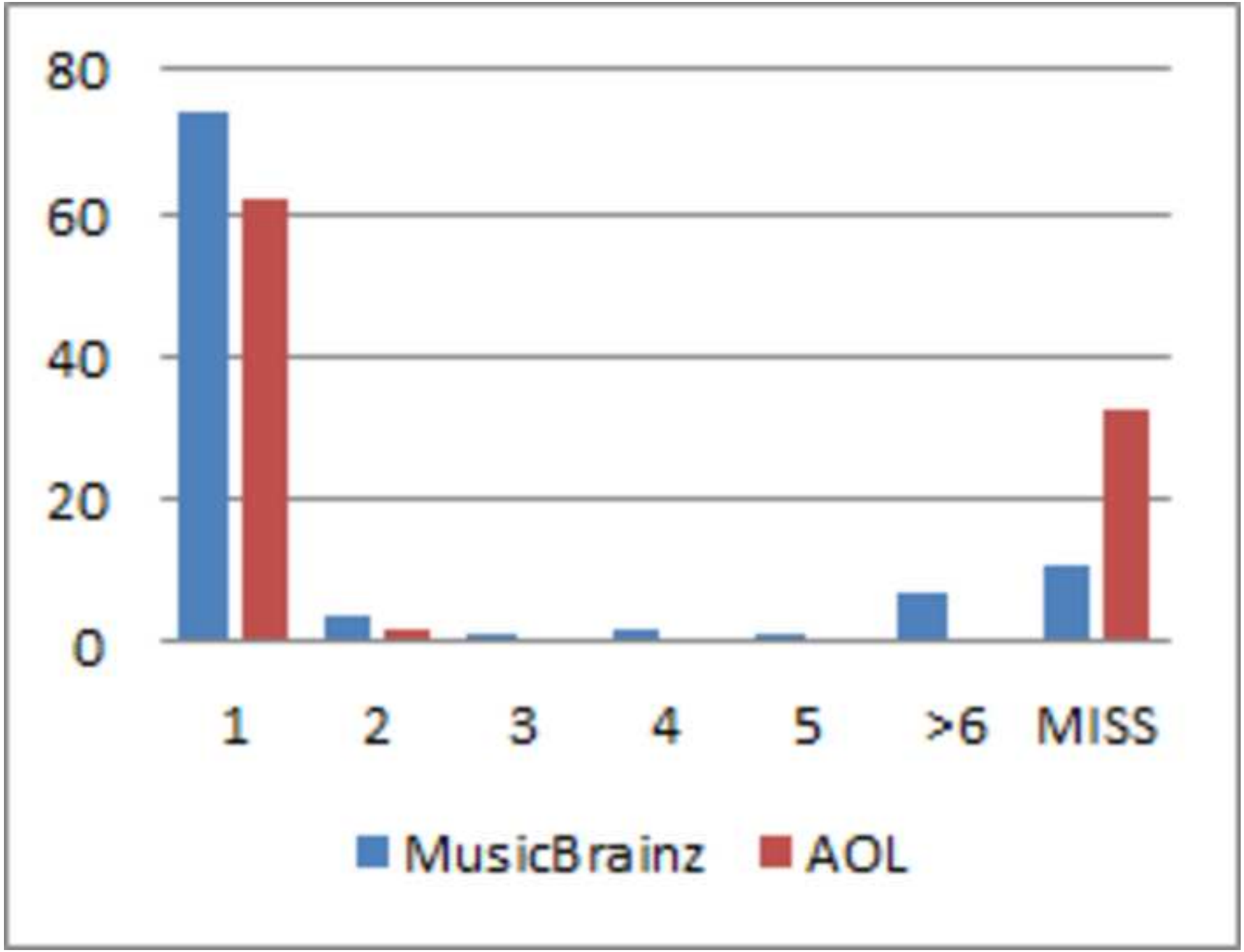
Figure

Click here to download high resolution image



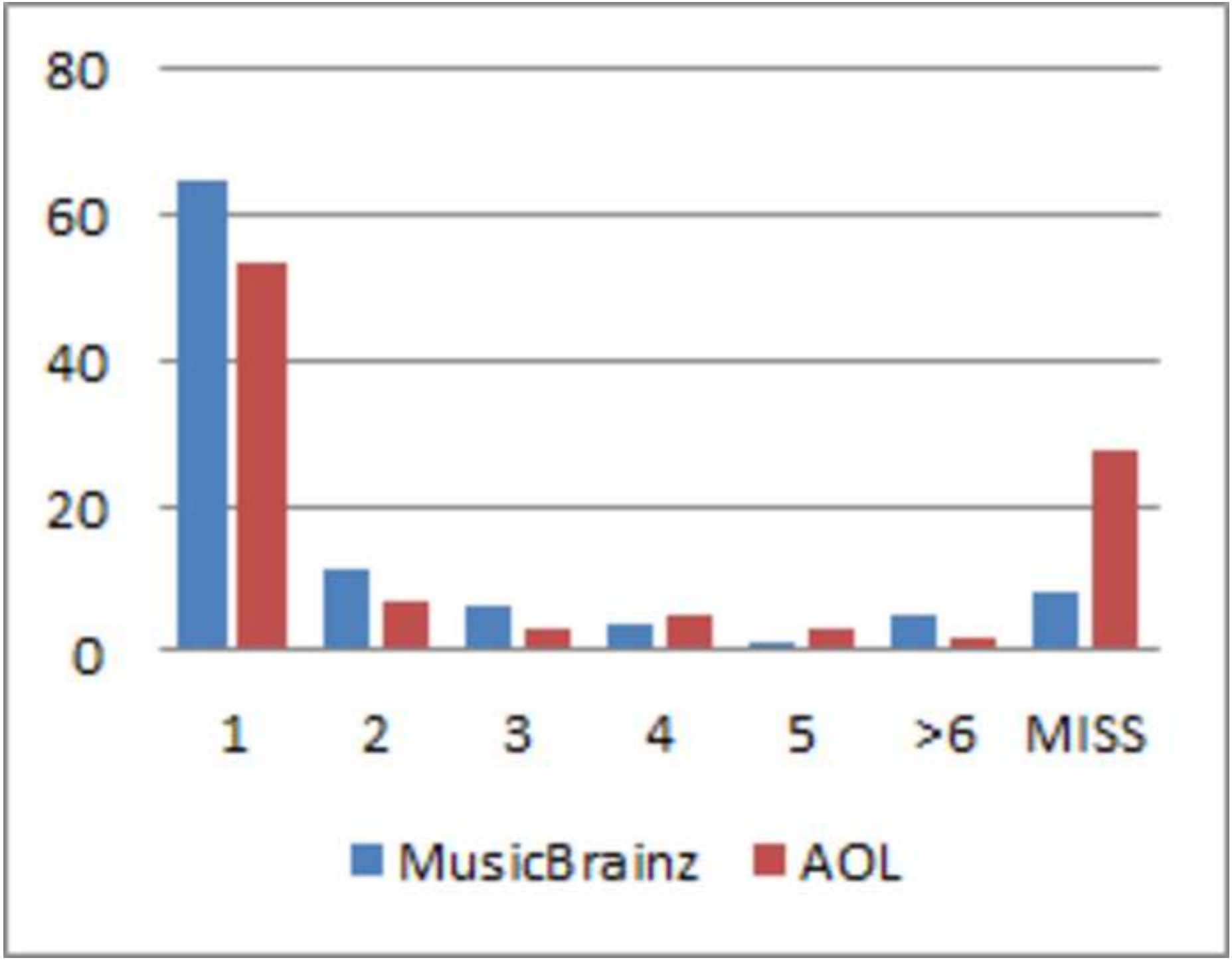
Figure

Click here to download high resolution image



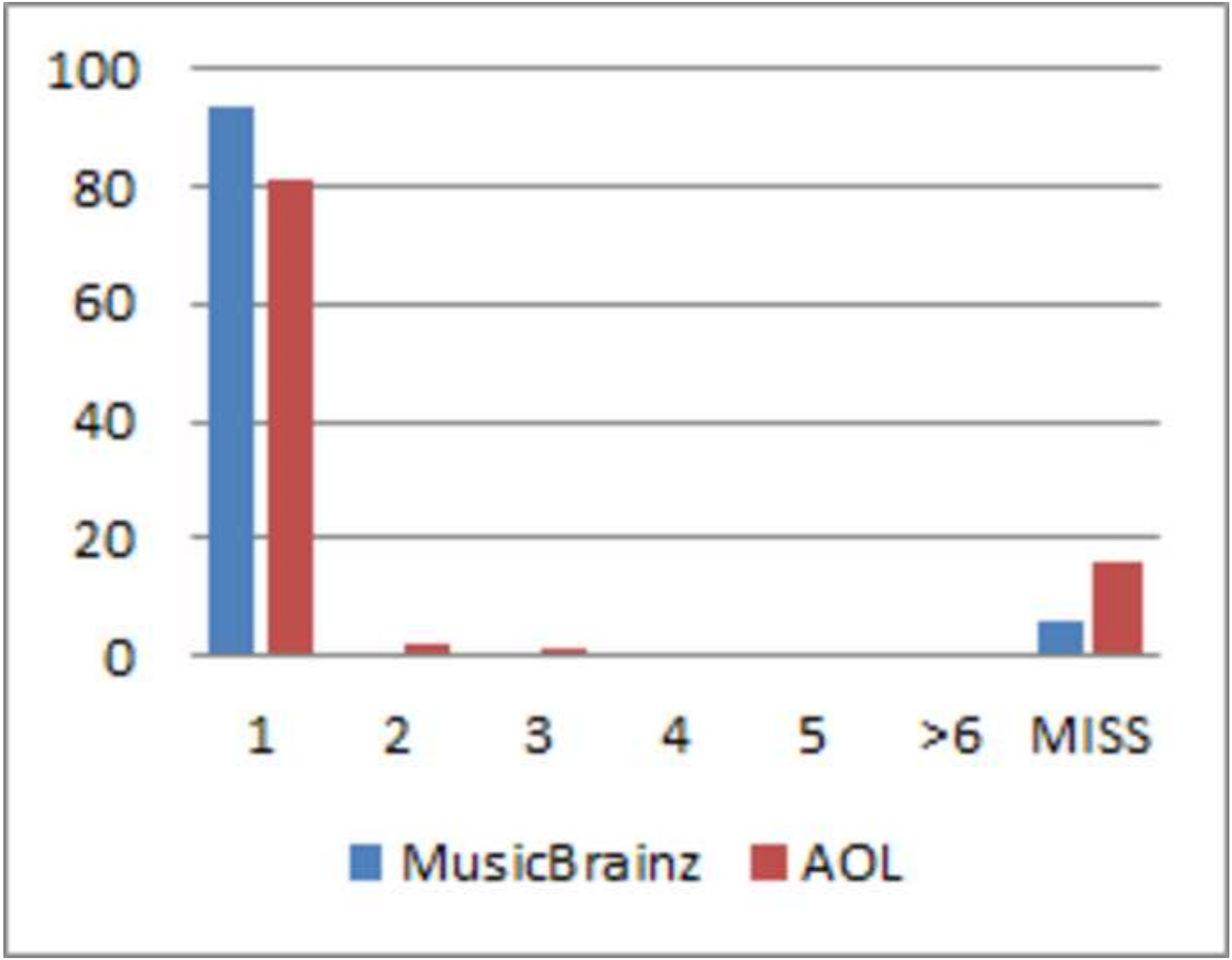
Figure

Click here to download high resolution image



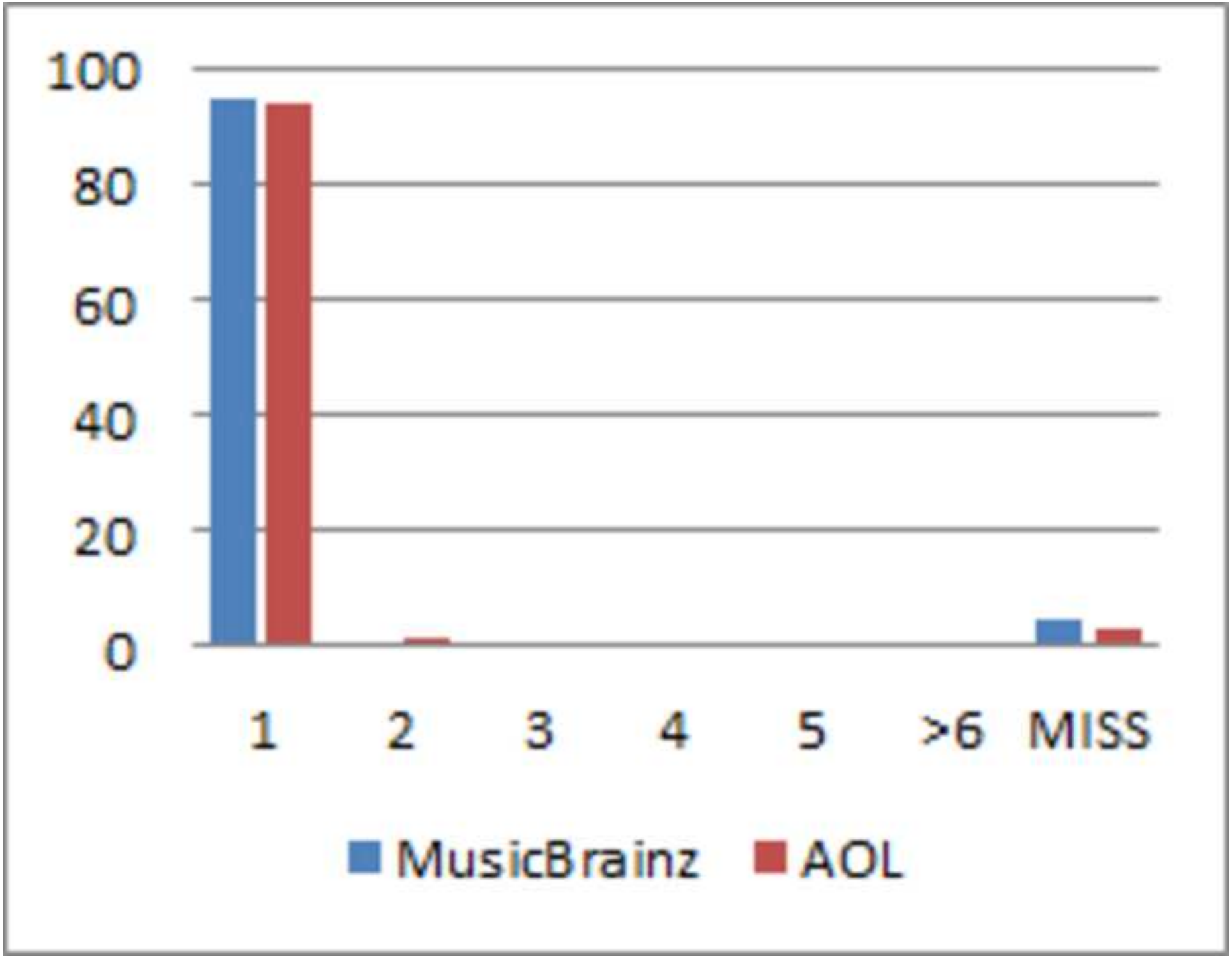
Figure

Click here to download high resolution image



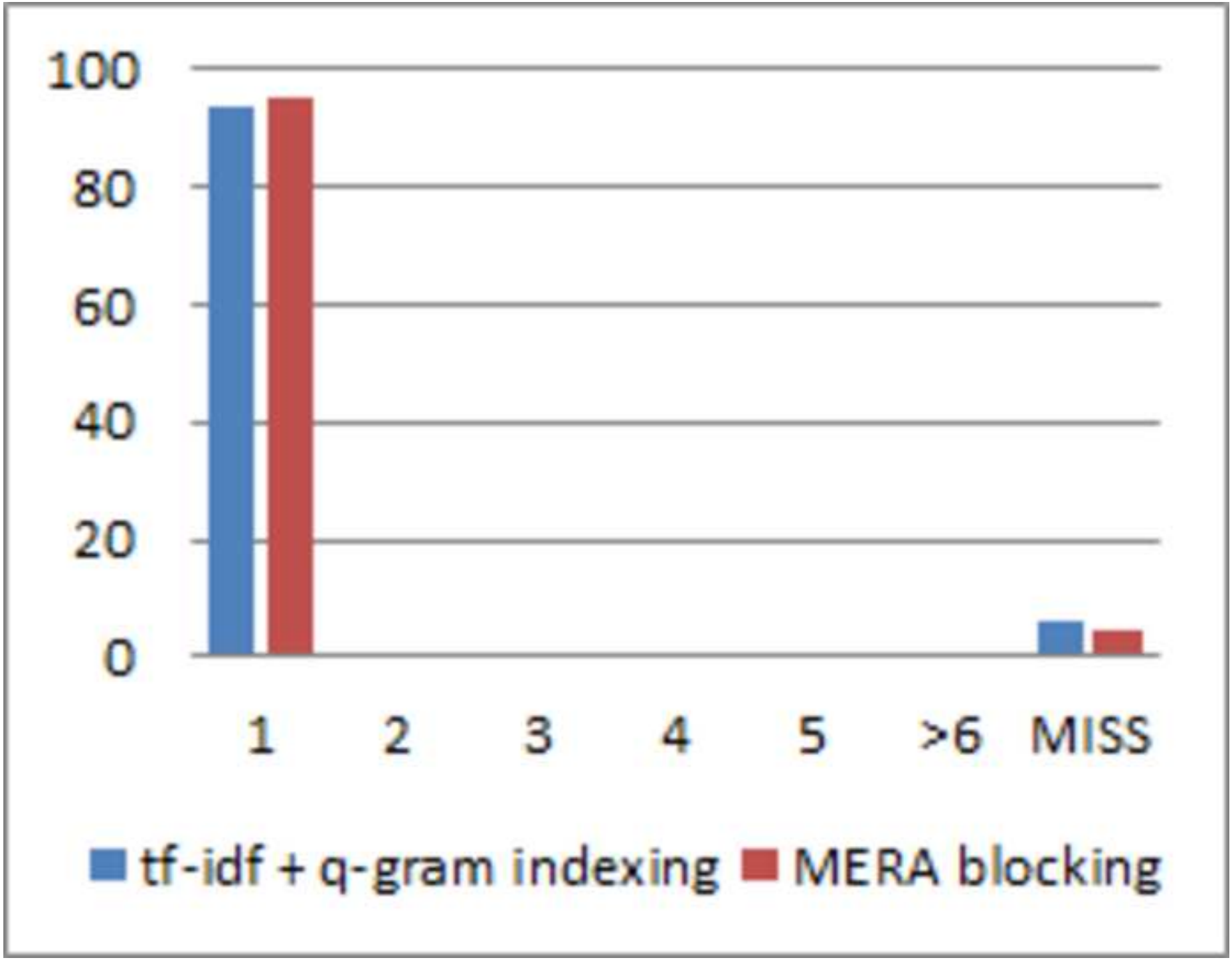
Figure

Click here to download high resolution image



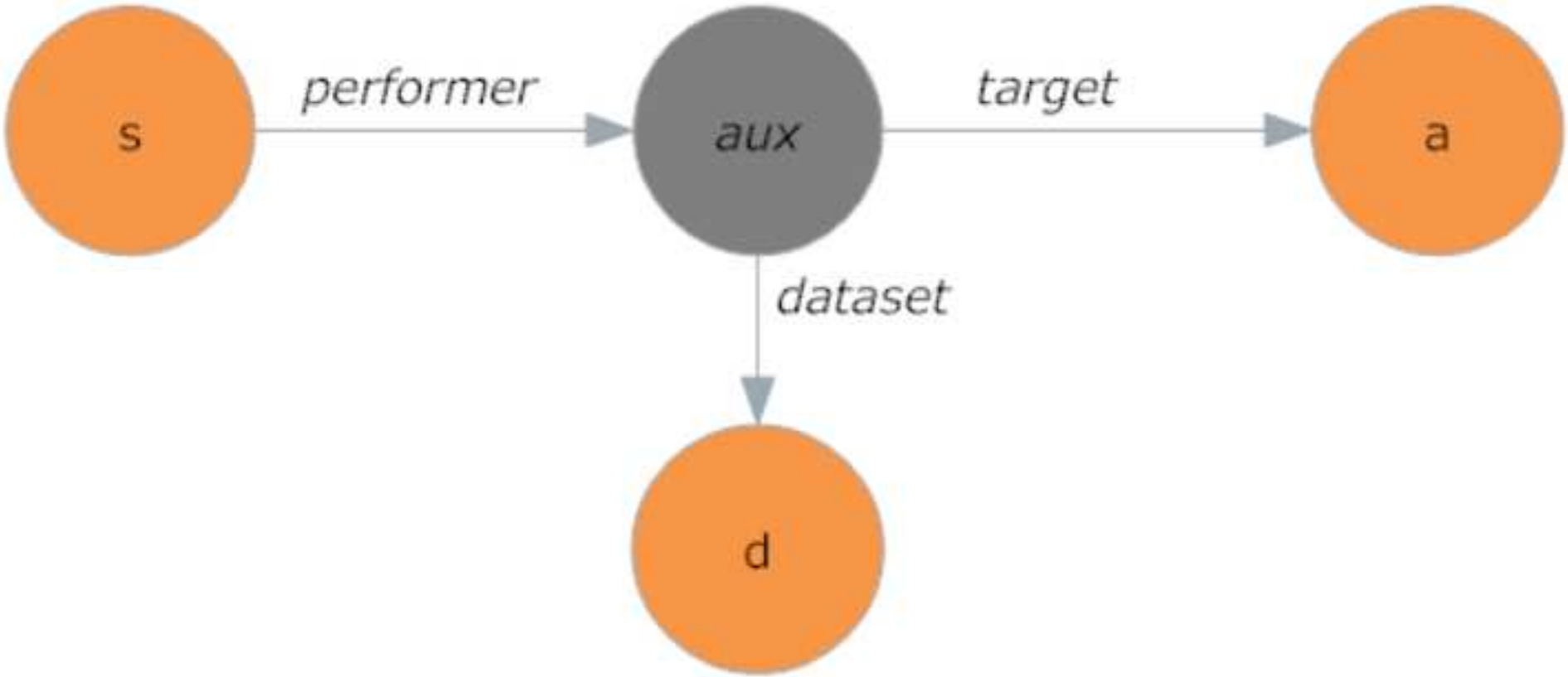
Figure

Click here to download high resolution image



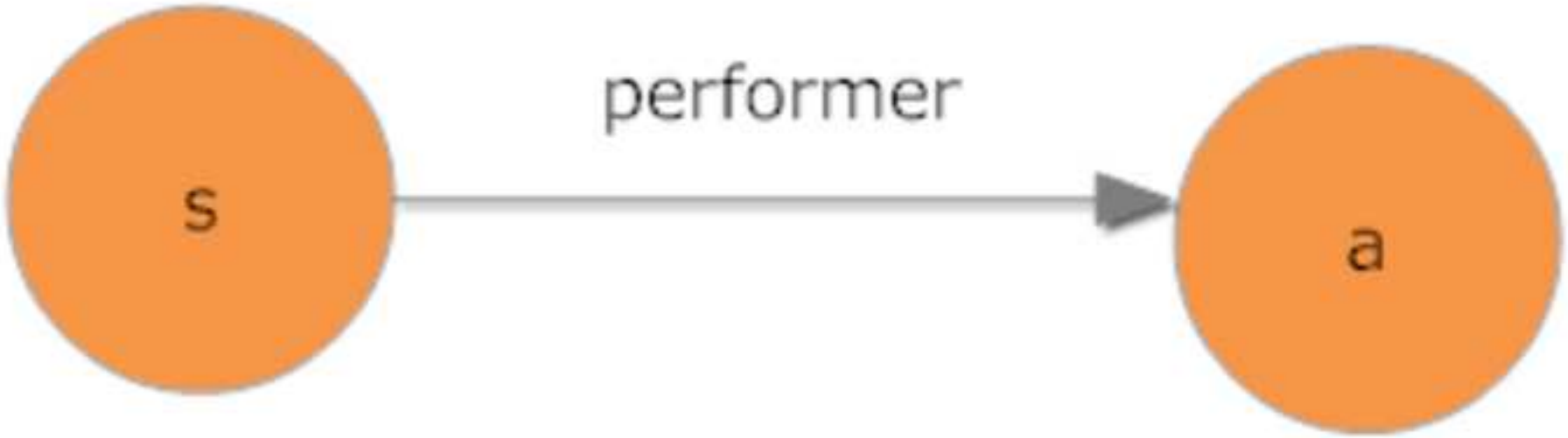
Figure

Click here to download high resolution image



Figure

Click here to download high resolution image



Figure

Click here to download high resolution image