Universidad CENFOTEC

Maestría Profesional en Tecnologías de Bases de Datos

Documento Final de Proyecto de Investigación Aplicada 2

Valoración de la eficiencia y eficacia de algoritmo GDS para ubicación espacial de nombres personales como apoyo a la detección de duplicados ingresados con un teclado QWERTY Latinoamericano

Ing. Keyla Alvarado Cantillano

Ing. Rigoberto Carvajal Pérez

Abril, 2015

Declaratoria de Derechos de autor

Declaramos que el presente proyecto de investigación ha sido realizado en su totalidad por los autores Keyla Alvarado Cantillano y Rigoberto Carvajal Pérez, basados en literatura referente al tema y conocimiento propio de los autores.

En los casos en que se ha hecho referencia a la literatura encontrada, se ha procedido a indicar las fuentes utilizadas mediante citas bibliográficas.

Dedicatoria

A Dios por darme la oportunidad, fortaleza y perseverancia para culminar esta maestría. A mis tres tesoros mi esposo Carlos Redondo y mis hijos Carlos y Sara por creer en mí, por ser mi inspiración y fuerza en todo momento.

Keyla Alvarado Cantillano.

A Dios, porque todo lo que hago lo hago como para Él, ya sea en mi trabajo o estudios. Él me ha dado todo lo que tengo, las oportunidades, la experiencia, las habilidades, los recursos y lo mejor de todo: mi salvación.

Rigoberto Carvajal Pérez

Agradecimientos

Agradezco a Dios quien es mi guía, mi fuerza, como en todo momento de mi vida, una vez más me acompañó y ayudó a vencer los obstáculos que se presentaron durante estos 2 años y medio, también agradezco el haberme permitido conocer a seres humanos maravillosos en este lapso. Gracias Jesús, eres mi ejemplo a seguir. Al Espíritu Santo por iluminar mi camino.

Agradezco especialmente a mi querido esposo Carlos por su apoyo incondicional, sacrificio y entrega, por creer en mí, fuiste quien me impulsó a llevar a cabo este proyecto. Gracias a mis hijos Carlos y Sarita, sé que en muchas ocasiones el tiempo en familia fue sacrificado por mis obligaciones relacionadas a este proyecto, a los tres gracias por darme la fuerza y motivación para seguir adelante.

Gracias a mi querida madre, por enseñarme con su ejemplo a luchar y dar lo mejor en todo momento, a mis hermanas Judy y Verónica por el apoyo brindado.

Mi más sincera gratitud para mi suegro Carlos Redondo por su constante ayuda y disposición en todo momento que lo necesité.

Gracias a nuestro tutor y profesor Luis Naranjo, por compartir con nosotros su conocimiento, por su disponibilidad y entrega, por creer en este proyecto, por la exigencia académica solicitada a lo largo de este proyecto, por su motivación y ante todo por su calidad humana, le admiro y le estaremos siempre agradecidos.

Agradecemos a los señores Pedro Méndez y Arnoldo Muller por sus valiosos aportes.

Gracias a mi compañero Rigoberto Carvajal, por su entrega, dedicación, por tratar siempre de dar lo mejor, fue un gusto trabajar con usted no solamente en el presente proyecto sino a lo largo de toda la maestría, que el señor le bendiga a usted y su esposa.

Keyla Alvarado Cantillano.

Agradezco a Dios por haberme respaldado en este proyecto profesional con su fuerza y sabiduría.

Gracias a mi amada esposa Ericka Martinez,Quirós, por apoyarme en esta meta, ayudarme a conseguirla, por ser paciente con el tiempo que tuvimos que sacrificar juntos, y por todo su amor.

Gracias a mis padres Rigoberto Carvajal Solís y Esmeralda Pérez Torrentes, por haber trabajado bastante para darme el regalo de la educación, enseñarme el valor de las cosas y formarme como una persona responsable.

Gracias a mis pastores Carlos Cunningham y Karla Molina, quienes también me apoyaron en este proyecto y me han guiado en mi crecimiento espiritual.

Gracias a mi equipo de líderes del Ministerio de jóvenes de la Iglesia Jesucristo es el Señor en Calle Blancos, por su buen trabajo, compromiso y respaldarme en varias tareas muchas veces mientras duró este proyecto.

Gracias a nuestro tutor Luis Naranjo por todo su interés en el proyecto, por compartir con nosotros su conocimiento y pasión por la investigación los cuales han influenciado para siempre nuestras vidas profesionales.

Gracias a mi compañera Keyla Alvarado por todo su compromiso y excelente trabajo, juntos hicimos un gran equipo y un excelente proyecto.

Rigoberto Carvajal Pérez

Resumen ejecutivo

Una de las mayores oportunidades de mejora que se encuentra hoy en día en las empresas está relacionada con la calidad de los datos. La mayoría de técnicas, herramientas y algoritmos que se encuentran han sido desarrollados para el manejo de palabras de uso cotidiano y nombres de personas.

La corrección de nombres de personas, dada la cantidad de variantes que pueden existir, no ha sido una tarea fácil y esta genera muchos problemas, los cuales pueden limitar la toma de decisiones, incluso la creación de nuevas aplicaciones que permitan agilizar los procesos tanto internos como externos en las organizaciones.

En el siguiente trabajo se presenta un nuevo algoritmo llamado GDS (Geometric Distance for Strings), que ubica espacialmente cadenas de texto en un espacio tridimensional, predefiniendo clústeres de valores similares. Al combinarse GDS con la función de similitud SIMIL se convierte en una solución efectiva y eficiente para la detección de datos duplicados, que representan nombres de personas.

Se realizaron evaluaciones con datos reales y una comparación con algoritmos existentes para validar los resultados. El algoritmo ha sido desarrollado en código abierto, con el fin de que se puedan realizar aportes futuros de cualquier otra persona que esté interesada en el tema.

Palabras claves: GDS, Geometric Distance for Strings, Spatial location for strings,algoritmo híbrido, deduplicación, calidad de datos, vecino más cercano.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

1.1 Antecedentes del problema

Durante los últimos años se ha observado una era donde las empresas se han dedicado a maximizar el uso de los sistemas operacionales y acumular muchos datos en numerosas fuentes; sin embargo, el valor de esos datos aún no es claro para muchas organizaciones, y las que ya lo han descubierto lo han explotado por medio de las herramientas tecnológicas para inteligencia de negocio. Es, entonces, cuando la tarea de integrar todos estos datos se convierte en un tema importante ya que la calidad de los datos, en muchos casos, no es la adecuada, y en ese momento se descubre qué tantos problemas existen al respecto, por ejemplo: duplicidad de datos, incompletitud, inconsistencia, valores indeterminados o incorrectos.

1.2 Hipótesis

Es posible utilizar una función tal que dada una cadena de texto que represente un nombre de persona, pueda ubicarla en un espacio N-dimensional de manera que, si se aplica a un conjunto de cadenas de texto, pueda predefinir los clústeres de valores similares, reduciendo significativamente la cantidad de comparaciones que se llevan a cabo cuando se aplica un algoritmo de vecino más cercano para la deduplicación de datos con el propósito de mejorar su eficiencia y mantener, a su vez, un nivel de eficacia satisfactorio.

1.3 Justificación

El problema de la duplicación de datos es muy recurrente en las bases de datos y, si bien esto genera errores en los sistemas operacionales, estos no son tan evidentes, en muchos casos son visibles a la hora de hacer análisis de los datos, donde si no se observan con claridad se tomarán como referencia gráficos y reportes inconsistentes que pueden llevar a la toma de malas decisiones y, consecuentemente, a la pérdida de la confianza total de los sistemas para toma de

decisiones, llevando a la final decisión de desechar todo un sistema e infraestructura de alto costo. El Data Warehouse Institute estima que el costo de los datos de clientes de baja calidad es, aproximadamente, de $ 611 mil millones al año y las principales causas comúnmente asociadas con datos de baja calidad son: los errores de entrada de datos, reglas de negocio mal aplicadas, registros duplicados y la falta de valores o datos incorrectos (Eckerson, 2002). En un estudio reciente se estima que la mala calidad de los datos cuesta a una organización en promedio $ 13.5 millones por año (Gartner, 2015), la misma firma define “datos de mala calidad” o “defectuosos” los datos inexactos, incompletos o duplicados (Gartner, 2007).

Esta investigación está enfocada en aportar un mejoramiento en la calidad de los datos mediante un proceso eficaz y eficiente para apoyar la detección de registros duplicados, con especial énfasis en los registros duplicados que representan a personas.

1.4 Viabilidad

1.4.1 Factibilidad técnica

Se cuenta con una estación de trabajo suficiente para someter los algoritmos a prueba con conjuntos de decenas de millones de registros, con la siguiente especificación:

16GB (4x4GB) 1600MHz DDR3L

Intel® Core™ i7-4800MQ processor (Quad Core 2.70GHz, 3.7GHz Turbo, 6MB 47W, w/HD Graphics 4600)

1.5 TB Hybrid 2.5in, SATA3 with 8GB SSD Flash

1.4.2 Factibilidad económica

A continuación, en la tabla 1 se presenta la factibilidad económica que representa el desarrollo de esta investigación, el costo total será asumido por los investigadores (Estudiantes).

Tabla 1: Factibilidad económica anteproyecto.

	Factibilidad económica				
	Costos			**Tipo de Cambio**	540
Profesional	Unidad de medida	Costo unitario	Cantidad	Total dólares	Total colones
Profesional 1	Hora	40	250	10000	5400000
Profesional 2	Hora	40	250	10000	5400000
Costo total Investigación			500	16000	10800000

1.4.3 Factibilidad operacional

Se cuenta con experiencia en las herramientas y lenguajes que se utilizan para implementar y probar el algoritmo a desarrollar, como lo son ETL, conocimiento de múltiples motores de bases de datos y lenguajes como Java o C#.

1.5 Objetivos del proyecto

Para la especificación de objetivos se utiliza la taxonomía de Bloom, que permite mostrar en distintos niveles cognoscitivos, desde lo más simple hasta lo más complejo.

1.5.1 Objetivo general

Valorar la eficiencia y eficacia de algoritmo GDS para ubicación espacial de nombres personales ingresados con un teclado QWERTY latinoamericano**.**

1.5.2 Objetivos específicos

- Identificar los errores más comunes que generan datos duplicados.
- Seleccionar 4 algoritmos para la detección de valores duplicados.
- Formular un proceso de poda de comparaciones para la detección de valores duplicados.
- Aplicar sobre uno de los corpus los algoritmos seleccionados y el algoritmo propuesto, y solamente este último sobre otros dos corpus de distinto tamaño.

- Medir los resultados de los algoritmos seleccionados y el algoritmo propuesto.

1.6 Alcances y limitaciones

1.6.1 Alcances

El algoritmo generará clústeres de nombres personales similares, buscando que queden agrupados posibles registros duplicados. Para la generación de dichos clústeres se definirá un cubo que hará una función similar a un radio, el cual puede ser parametrizado por el usuario de acuerdo con el nivel de exactitud que se requiera. Por defecto, tiene un valor calculado obtenido de las pruebas sobre el corpus controlado. Dicho algoritmo podrá ser aplicado de forma genérica sobre campos de bases de datos que sean cadenas de caracteres de texto no numéricos, los cuales hayan sido ingresados utilizando teclados QWERTY ya sea en español latinoamericano o con la configuración requerida, será evaluado al menos con 3 conjuntos de datos obtenidos de la siguiente manera: dos procedentes de bases de datos de origen público y un último set de datos con errores generados de forma intencional.

1.6.2 Limitaciones

Para todo efecto, se tomarán en cuenta solamente los campos de tipo texto que corresponden a nombres propios de personas físicas, con un largo máximo de 60 caracteres; los valores numéricos o alfanuméricos serán considerados en futuras posibles ampliaciones del algoritmo.

Para efectos de prueba se generará un conjunto de datos con errores que son bien conocidos por los investigadores, con el fin de evaluar la calidad del algoritmo en comparación con otros.

2 Estado de la Cuestión

Una investigación parte de un nivel de conocimiento que puede servir de apoyo para formar criterio y confirmar el nivel de información que existe alrededor de un tema; para ello, se requiere con anterioridad, mediante determinados procedimientos de trabajo, obtener información que ha sido propuesta por otros investigadores. A la búsqueda de lo que otros han producido como conocimiento sobre un objeto de trabajo o tema-problema, y al informe escrito que consuma esta indagación, se les considera como los elementos que configuran la realización del estado de la cuestión (Gallardo, 1999).

Se han desarrollado diferentes métodos y algoritmos para encontrar de manera automática datos duplicados. Ha sido importante llevar a cabo una investigación sobre diferentes publicaciones y, de esta manera, ir profundizando en el tema y lograr obtener el conocimiento necesario para la adaptación o desarrollo del algoritmo. Se estudiaron 2 mecanismos utilizados para la detección de duplicados: uno de ellos mediante el uso de colisión de llaves; el otro, mediante funciones de distancia. Si bien es cierto el primero se ejecuta en menor tiempo, detecta una cantidad menor de errores de datos duplicados que la función de distancia.

En este apartado se describen la revisión sistemática empleada para la búsqueda de investigaciones, los métodos y los principales algoritmos existentes encontrados, relacionados con el manejo de datos duplicados sobre cadenas de caracteres.

2.1 Revisión sistemática

Formulación de la cuestión

Enfoque de la cuestión: identificar los problemas existentes en cuanto a la duplicación de datos de tipo cadena de caracteres y algoritmos que ayuden a resolver esta situación.

Calidad y amplitud de la cuestión

Problema: La calidad de datos es un problema recurrente en todos los sistemas de información, uno de esos problemas es la existencia de registros en las bases de datos que representan a un mismo objeto pero esto no se evidencia en la manera como están registrados.

Cuestión: ¿Qué retos y algoritmos existen para resolver el problema de duplicidad de datos de manera eficaz y eficiente?

Palabras claves, sinónimos y traducciones:

Calidad de datos: Data Quality

Duplicidad de datos: Duplicación de datos, repetición de datos, duplicated data

Homologación de registros: Data Deduplication data Algorithms: Algoritmos para Eliminación de Duplicados

Intervención: Evaluación e identificación de algoritmos para la solución de duplicidad de datos de nombres de personas

Control: No se tienen datos iniciales para la revisión sistemática

Efecto: Identificación de algoritmos existentes para encontrar oportunidades de mejoría

Medida de resultado: Número de algoritmos existentes para la deduplicación de datos, específicamente para el manejo de nombres de personas o cadenas de caracteres, su nivel de eficiencia, eficacia, error y complejidad en tiempo y espacio.

Población: Publicaciones sobre el manejo de duplicidad de datos mediante algoritmos de colisión de llave, funciones de distancia y/o algoritmos híbridos.

Aplicación: Desarrolladores en Inteligencia de Negocio. Analistas de datos.

Diseño experimental: Se utilizará el método estadístico llamado muestreo intencional para la evaluación de uno de los experimentos.

Selección de fuentes

Definir criterio de selección de fuentes:

Acceso a las bibliotecas de artículos Web; existencia de mecanismo de búsqueda usando palabras y frases claves y sugerencias de expertos
Entrevistas con expertos
Lenguaje: Inglés/Español

Identificación de fuentes:

Métodos de búsqueda: Motores de búsqueda en la Web. Contacto con expertos

Cadena de búsqueda:

- ("Data Quality" OR Data deduplication" OR "Duplicated data") AND "personal names" AND algorithms

- ("Calidad de datos" OR "Eliminación de duplicados" OR " Deduplicación de datos") AND "Nombres personales" AND algoritmos)
- "Clustering Strings": “Agrupamiento de Cadenas de texto”
- “Nearest neighbor pruning”: “poda del vecino más cercano”
- “Spatial hashing”: “Hashing Espacial”
- “String Distances”: “Distancia entre cadenas de texto”

Lista de recursos:

IEEE xplore DIGITAL LIBRARY, EBSCO Discovery Service, Pro Quest, Science Direct, Web of Science, ACM, IJERT, Google Académico.

Selección de estudios

Definición criterios de inclusión y exclusión:

Se tomarán en cuenta los estudios donde se expliquen y se apliquen algoritmos para la deduplicación de nombres de personas o cadenas de caracteres.

Se excluirán los estudios que están centralizados en la presentación de un producto de *software* comercial para calidad de datos.

Definición tipo de estudios: Cualquier tipo de estudio que cumpla con los criterios definidos anteriormente.

Procedimiento para selección de estudios: Se ejecutarán los distintos criterios de búsqueda definidos en las fuentes seleccionadas, el primer filtro para los estudios será por medio de la lectura del resumen ejecutivo e introducción; luego, se refina la lista de estudios mediante la lectura completa del texto.

2.2 Investigaciones

En el presente apartado se describen las investigaciones encontradas de interés relacionadas con el tema propuesto en este trabajo, estas investigaciones utilizan diferentes métodos para realizar las búsquedas y comparaciones entre cadenas de caracteres, para la detección de datos duplicados.

(Amón, Moreno, & Echeverri, 2012)**,** en su investigación llamada *Algoritmo Fonético Para Detección De Cadenas De Texto Duplicadas En El Idioma Español*, presentaron un algoritmo el cual nombraron PhoneticSpanish, de codificación fonética, similar al algoritmo Soundex con algunas adaptaciones para el idioma español. Este algoritmo ha sido probado mediante datos generados de forma aleatoria, se realizaron comparaciones con nueve técnicas convencionales para la detección de duplicados. Los resultados mostraron que fue muy eficaz utilizando como métrica la función de discernibilidad (veéase definición más adelante).

Otras investigaciones estudiadas utilizan diferentes métodos para detectar duplicados, estas utilizan combinaciones de distintas técnicas por ejemplo (Christen, 2006) presentó en su investigación una propuesta donde combinando algoritmos, tanto de codificación fonética (Soundex, Phonix) como de patrones de búsqueda. (Levenshtein distance) desarrolló algoritmos híbridos los cuales llamó Editex y SyllAlign, y llevó a cabo experimentos para valorar el desempeño de estos, encontrando pares de cadenas de caracteres similares.

Para realizar el análisis del experimento, aplicó sobre un mismo corpus algoritmos fonéticos y de patrones de búsqueda de forma individual, también realizó el experimento con los algoritmos híbridos; luego, comparó los resultados obtenidos, y se encontró que los algoritmos híbridos no presentaron el mejor desempeño sino, más bien, una puntuación intermedia. Los algoritmos con técnicas de codificación fonética fueron superados por los algoritmos que utilizan

patrones de búsqueda, los experimentos mostraron que no existe una técnica mejor que otra, sino que esto depende mucho del conjunto de datos que se desea analizar y los tipos de errores que estos contienen.

Dado el tamaño y complejidad de las bases de datos, surge la necesidad de encontrar técnicas que permitan acelerar la comparación y detección de duplicados en grandes cantidades de registros. (Christen, 2012), en otra investigación muestra cómo mediante la utilización de diferentes técnicas de indexación se puede mejorar la eficiencia de búsquedas de pares similares en una base de datos de gran tamaño. Todas las técnicas de índices estudiadas en su investigación se basan, primeramente, en una llave llamada Blocking Key Value (BKV). Algunas de estas técnicas de indexación vienen siendo utilizadas desde los años sesentas tales como el bloqueo estándar o tradicional; otras más recientes como índices de ordenamiento de llaves basados en arreglos o índices invertidos fueron tomados en cuenta. También evaluó los índices basados en Q-gram y sufijos, Canopy Clustering utilizando umbrales o el vecino más cercano. Los resultados arrojados en los experimentos permitieron ver y comparar el rendimiento de las diferentes técnicas, midiendo la cantidad de pares encontrados con cada una de ellas, la calidad de los pares encontrados, así como el comportamiendo de los índices realizando modificaciones en parámetros tales como reducción del radio (RR) de cada clúster. Para medir el rendimiento, desempeño y cantidad de errores de estas técnicas fue muy importante la correcta generación de los BKV.

3 Marco teórico

El objetivo del presente capítulo es describir los conceptos relacionados con el tema a tratar tales como calidad de datos, duplicidad de datos, los diferentes tipos de errores por medio de los cuales se generan datos duplicados y algunos otros conceptos. También se describen y analiza la funcionalidad de los algoritmos más utilizados para realizar las búsquedas y detecciones de datos duplicados importantes para la propuesta en esta investigación.

3.1 La calidad de los datos

3.1.1 Calidad de los datos

No existe una definición simple para el concepto de "calidad de datos"; no obstante, los aspectos como exactitud (datos que están correctos o actualizados), consistencia (valores que no tienen conflictos entre sí), no duplicidad (datos no duplicados), integridad (datos que no carecen de referencias importantes de relación) completitud (inexistencia de datos faltantes o sin uso) son utilizados comúnmente para representarla. Según explicaba (Hernández M. , 2013), una de las definiciones más aceptadas para el término de calidad de datos es "Adecuación al uso"; por lo tanto, en la calidad está incluido no solamente que los datos tengan cero defectos sino que tengan un valor agregado para los usuarios a la hora de trabajar con ellos.

Para (Tayi & Ballou, 1998), un dato puede considerarse de calidad si cumple con su propósito y es apto para utilizarse; es decir, el dato se analiza desde un enfoque de los aspectos mencionados anteriormente, pero referenciados en un contexto de usuario más que en un contexto puramente técnico.

3.1.2 Duplicidad de datos

La duplicidad de datos se presenta cuando, en una misma entidad del mundo real, se almacenan, más de una vez, registros los cuales no son idénticos. El proceso para detectar este tipo de problemas en el área de bases de datos se conoce como data deduplication o instance identification. Este proceso fue identificado inicialmente por Halbert L Dunn, en 1946. Algunos fundamentos probabilísticos y mejoras fueron desarrollados posteriormente. El proceso funciona de la siguiente manera: dado un conjunto *R* de registros: a) se define un umbral de diferenciación; b) se compara cada registro con los demás registros; y c) si la similitud entre una pareja de registros es mayor o igual que el umbral, se supone que son duplicados y se considera que son representaciones de una misma entidad del mundo real (Amón, Moreno, & Echeverri, 2012).

3.1.3 Tipos de errores que generan datos duplicados en cadenas de caracteres

Un problema habitual de la recuperación de cadenas de caracteres es la determinación de todas las formas variantes en el momento en que estas cadenas se introducen en la base de datos. Las variantes se producen por distintas causas como tales como errores ortográficos, fonéticos o tipográficos (que darían lugar a omisiones, inserciones o sustituciones de caracteres en las cadenas), uso incorrecto de mayúsculas, errores de acentuación, o distinta distribución de los componentes del nombre propio.

Los errores tipográficos (también conocidos como *fat-fingering*) se producen cuando una letra se escribe accidentalmente por otra debido a la posición y rapidez con que fue digitada, por ejemplo, las letras “i” y “o” que son contiguas en los teclados QWERTY. Un caso puede darse cuando se escribe "Carlis" al tratar de escribir "Carlos". Se basan en el supuesto de que quien escribe sabe cómo deletrear la palabra, pero puede haber escrito la palabra rápidamente. Los

errores cognitivos se refieren a situaciones en las que quien escribe elige una ortografía incorrecta debido a la falta de conocimiento de la correcta. Un ejemplo sería la ortografía incorrecta de "Fernández" como "Fernándes". Errores fonéticos pueden ser considerados como un subconjunto de los errores cognitivos. Estos errores se producen cuando quien escribe sustituye letras en una palabra donde el sonido se cree erróneamente que es correcto, que podría conducir a una falta de ortografía o bien la cadena podría existir en más de una forma tal como “Karla” y “Carla”.

3.2 Conceptos importantes

3.2.1 Similitud de datos

Se considera que existe similitud de datos cuando estos tienen ciertas características parecidas tales como sonido similar o forma de los caracteres. En el caso de las cadenas de texto, específicamente nombres personales, existen diferentes métodos para calcular la distancia entre un par de ellos. Entre menor distancia exista, los nombres más parecidos son, de lo contrario entre mayor distancia más disímiles son.

3.2.2 Función de discernibilidad

Esta función es una métrica que ha sido usada en trabajos comparativos para determinar cuán eficaces son las funciones de similitud identificando las entradas que realmente corresponden a un mismo objeto, incorpora los cuatro elementos tradicionales de una matriz de confusión: aciertos, desaciertos, falsos positivos y falsos negativos.

3.2.3 Valores llave para generación de bloqueos

Se conocen como BKV’s. Estos valores están basados en un atributo o varios, sobre los cuales se realizan transformaciones mediante la aplicación de un algoritmo fonético sobre alguno

(s) de los atributos que conforman la llave más la concatenación de letras o números de los otros atributos elegidos. El criterio para elegir cuáles atributos tomar en cuenta para la generación de esta llave es muy importante, no se recomienda elegir atributos donde los valores almacenados sean muy repetidos o tengan muchas variaciones, además de validar la calidad de los datos almacenados en dichos campos, ya que una llave errónea, no permitiría que los procesos que utilicen estas llaves puedan arrojar resultados confiables.

3.2.4 Agrupación de datos (Cluster)

Significa agrupación y representa un conjunto de objetos que tienen similitud entre ellos y disimilitud con los objetos que pertenecen a otra agrupación. Se pueden considerar como la búsqueda automática de una estructura o de una clasificación en una colección de datos definidos en función de unos objetos de referencia conocidos como centroides.

3.2.5 Clustering

Es el proceso mediante el cual se agrupan los datos en clúster de manera que los objetos de un clúster tienen una similitud alta entre ellos, y sean muy diferentes con objetos de otros clúster.

Características importantes a destacar:

- Tiene la capacidad de manejar diferentes tipos de atributos: numéricos, binarios, nominales, ordinales, entre otros.
- Capacidad de agregar restricciones.
- Pueden funcionar eficientemente con alta dimensionalidad.
- Requerimientos mínimos para especificar parámetros, como el número de clúster.
- No tiene una dependencia del orden de los datos.
- Permite que los clúster sean interpretables y utilizables.

3.3 Algoritmos

Para resolver un problema dado es necesario desarrollar una estrategia la cual, mediante una serie de procesos permita dar la solución que se requiere, (Storti, D'Elía, Paz, Dalcín, & Pucheta, 2012) exponen que una forma abstracta de plantear una estrategia es en la forma de un "algoritmo"; es decir, una secuencia de instrucciones cada una de las cuales representa una tarea bien definida y puede ser llevada a cabo en una cantidad finita de tiempo y con un número finito de recursos computacionales. Un requerimiento fundamental es que el algoritmo debe terminar en un número finito de pasos, de esta manera él mismo puede ser usado como una instrucción en otro algoritmo más complejo.

Los algoritmos que se detallan a continuación han sido algunas de las soluciones que se han desarrollado para solventar problemas de limpieza de datos, duplicación de datos y realizar comparaciones para encontrar similitud en cadenas de texto, estas comparaciones son muy útiles en diferentes áreas como la Bioinformática (aplicación de tecnología de computadores a la gestión y análisis de datos biológicos), Minería de Datos e Inteligencia Artificial.

Algunos motores de bases de datos, aplicaciones de *software* especializadas en el manejo, integración y limpieza de datos, correctores ortográficos, e incluso lenguajes de programación ofrecen funcionalidades que utilizan algoritmos ya sean fonéticos o no fonéticos. Estos también han sido utilizados en aplicaciones para correcciones ortográficas, interfaces de búsquedas y sitios de genealogía. Una de las herramientas encontradas durante la presente investigación que utiliza varios de estos algoritmos es Open Refine, una aplicación Web para limpieza de datos que ofrece múltiples funcionalidades, utiliza Metaphone3 para realizar comparaciones fonéticas, también Fingerprint, Distancia de Levenshtein, y Ngram, algoritmos de bloqueo y vecino más cercano (OpenRefine, 2014).

3.3.1 Algoritmos de colisión de llave

Los algoritmos que funcionan con colisión de llave toman una cadena de entrada S, un conjunto de cadenas de texto a los cuales les aplican una serie de transformaciones a cada uno de sus elementos generando una cadena de salida K, llamada código o llave, de manera que a las cadenas que generan la misma llave, se les considera como duplicados. Su mayor ventaja es que son de complejidad computacional lineal.

3.3.1.1 Algoritmos fonéticos

La existencia de los algoritmos fonéticos obedece a la necesidad de recuperar información que tiene una semejanza sonora; y cuya representación a través de la palabra escrita puede diferir de su pronunciación (Gonzales-Cam, C. 2008). La codificación fonética permite reducir a una forma común las palabras que son similares en cuanto a su pronunciación, de esta forma se facilita la comparación de una cadena de caracteres con otra, debido a que se comparan las llaves generadas en lugar de la palabra completa.

Los algoritmos fonéticos son muy utilizados por diferentes herramientas; por ejemplo, correctores ortográficos como GNU Aspell, interfaces de búsqueda, *software* para deduplicación de datos, sitios de genealogía, aplicaciones para limpieza de datos (GNU Aspell, 2014). Para programas en Java, existen librerías de utilidad de código abierto que implementan diferentes algoritmos fonéticos, la compañía Antropomorphic Software LLC comercializa implementaciones de Metaphone3 en varios idiomas como Inglés, Español y Alemán e incluso otros lenguajes de programación como C#, PHP, Perl y PL/SQL.

Sin embargo, la principal limitación de estos algoritmos es que son dependientes del idioma utilizado, lo que hace necesaria la realización de modificaciones de acuerdo al idioma que se va a emplear.

Soundex: Fue desarrollado y patentado por Odell y Russell, en 1918. Reduce particularmente apellidos ingleses a un código de cuatro caracteres. El primer carácter es una letra mayúscula y los tres restantes son dígitos. (Knuth, 1973) describe el procedimiento utilizado por Soundex por medio de una función que consiste en: a) la conversión de caracteres a un código fonético; b) un algoritmo que sustituye todos los caracteres, excepto el primero, por su correspondiente código fonético; c) la eliminación de cualquier repetición consecutiva de caracteres; y d) la devolución únicamente de los primeros cuatro caracteres de la cadena resultante.

Muchos de los motores de bases de datos tales como Microsoft SQL Server y Oracle utilizan funciones que se basan en el algoritmo Soundex para realizar comparaciones fonéticas de nombres o palabras de acuerdo al sonido que emiten. En el caso de Microsoft las funciones (soundex y differences) (Microsoft, 2014), Oracle mediante la función llamada soundex (Oracle, 2014). IBM en su Gestor de datos maestros IBM InfoSphere Master Data Management utiliza el algoritmo Soundex (IBM, 2014).

Una de las desventajas encontradas con este algoritmo para utilizarlo con el idioma español se debe a la limitación de 4 caracteres para construir el código, lo que no le permite ser eficiente para detectar errores ortográficos comunes del español.

Metaphone

Fue desarrollado por Lawrence Phillips, y publicado en Computer Language*, Vol. 7, No. 12 (December), 1990, como parte de una clase de algoritmo llamado phonetic matching o phonetic encoding. Este algoritmo utiliza reglas de codificación mucho más extensas que su predecesor Soundex. Es un sistema de codificación fonética que elimina las vocales de las palabras, manteniéndolas solamente si son la primera letra de la palabra; también, elimina la

repetición de caracteres. Por lo tanto, la cadena generada es un conjunto de caracteres no repetidos que representan como se pronuncia una palabra en el idioma inglés, después de aplicar el conjunto de reglas definidas en el algoritmo.

Al día de hoy existen 3 versiones de este algoritmo, cada una mejora las reglas de codificación de la versión anterior por lo que aumenta la precisión de codificación. La versión más reciente es el Metaphone3, el porcentaje de precisión es de un 98%, para el idioma inglés. En esta última versión se encuentra disponible un producto comercial vendido como código para el idioma español y alemán.

Double Metaphone

Este algoritmo es una versión mejorada del Metaphone que elimina algunas ambigüedades del algoritmo original.

Los algoritmos Metaphone, según la investigación realizada, y como se ha venido comentando durante todo este apartado, son muy utilizados en la mayoría de aplicaciones, sin embargo en motores de bases de datos no se encontró que ninguno haya implementado el algoritmo como parte de la funcionalidad que ofrece. Si se encontró el código del algoritmo en diferentes lenguajes de programación (Aspell, 2014).

Phonix

Este algoritmo viene a resolver el problema que presentaba Soundex en cuanto a la capacidad de establecer algún tipo de ordenación entre las cadenas similares. Es una variante de Soundex cuyo algoritmo es más complejo. Se basa en la sustitución de todos los caracteres menos el primero por valores numéricos, además elimina de todas las apariciones del valor ‘0’. Phonix presenta algunas variaciones. Primero, realiza previamente unas 163 transformaciones de grupos de letras que normalizan las cadenas (por ejemplo, el carácter ‘X’ se transforma en

‘ECS’; además, si la primera letra es una vocal o la consonante ‘Y’ la transforma en ‘V’). La aportación más importante de este sistema de codificación es que computa los sonidos finales, y como consecuencia de esto es capaz de establecer tres rangos de similitud constituidos por palabras que concuerdan: en los sonidos finales, en los prefijos de los sonidos finales, o con sonidos finales distintos.

3.3.1.2 Algoritmos de colisión de llaves basados en los caracteres/*tokens*

Los algoritmos de este tipo han sido utilizados en el campo de la biología y sus necesidades bio-informáticas por ejemplo en el procesamiento de cadenas de ADN, ya que mediante un alfabeto de tan solo cuatro símbolos, se logra representar el código genético de los seres vivos y dado que las cadenas de ADN pueden diferir un poco unas de otras y ser lo suficientemente similares como para reportar una concordancia positiva importante para el estudio de las mismas, surge la necesidad de realizar comparaciones entre ellas (García, 2007).

Los algoritmos no Fonéticos han sido estudiados y utilizados en un gran número de disciplinas; por ejemplo, para el reconocimiento de patrones en el procesamiento digital de imágenes o musicología, también para la comparación de archivos, corrección ortográfica de textos y, sobre todo, en la minería de datos en las etapas relacionadas con la depuración y calidad de los datos para, de esta forma, obtener datos confiables que sean realmente útiles para la toma de decisiones, en el campo de la Inteligencia Artificial.

Fingerprint: Es un algoritmo muy sencillo y funciona para las deduplicaciones más estrictas, considera factores de duplicación como el orden y repetición de *tokens*, la capitalización, los acentos y uso o no de signos de puntuación, según como se explica en (OpenRefine, 2014) la transformación que se realiza es la siguiente:

1. Se quitan los espacios en blanco del inicio y el final

2. Se pasa toda la cadena a minúsculas
3. Se eliminan los signos de puntuación y de control
4. Se divide la cadena de texto en *tokens* separados por espacios en blanco.
5. Se ordenan los *tokens* y se elimina los duplicados
6. Se concatenan nuevamente los *tokens*
7. Se tranforman los caracteres occidentales extendidos (como los acentos especiales) a su representación ASCII.

El uso de este algoritmo fue encontrado en aplicaciones de limpieza y corrección de datos

N-Gram: Es un algoritmo que tiene las ventajas del Fingerprint y, además, captura errores provocados por confusiones en el orden de las letras. Según como se explica en (OpenRefine, 2014) la transformación que se aplica es la siguiente:

1. Se pasa toda la cadena a minúsculas.
2. Se eliminan los signos de puntuación y de control y espacios en blancos.
3. Se generan todos los n-gramas.
4. Se ordenan los n-gramas y se eliminan los duplicados.
5. Se concatenan los n-gramas ordenados.
6. Se normalizan los caracteres occidentales extendidos a su representación ASCII.

El uso de este algoritmo fue encontrado en aplicaciones de limpieza y corrección de datos.

3.3.2 **Algoritmo de Vecino más cercano**

Es muy sencillo de implementar, consiste en clasificar un objeto desconocido en la clase de su vecino más cercano, según una medida de disimilitud o distancia. La clasificación de un nuevo dato puede ser costosa computacionalmente cuando el conjunto de datos almacenado es

muy grande, debido a la necesidad de calcular los valores de proximidad del dato a clasificar y cada instancia de este conjunto. Por otra parte, la calidad de las clasificaciones depende del valor de K (cuántos vecinos se toman en cuenta para realizar la predicción), generalmente se determina de manera experimental.

Otro parámetro que puede influir en los resultados es la métrica de la distancia a emplear. Estos algoritmos son menos estrictos que los algoritmos por colisión de llave; por ello, puede capturar mayor cantidad de errores, pero resultan demasiado lentos porque requieren que todas las cadenas sean comparadas contra el resto de cadenas para poder calcular las distancias entre todos los elementos; para evitar esto, se utilizan mecanismos para podar toda esa cantidad de comparaciones tales como método de bloqueos. Este tipo de algoritmos es muy utilizado en la minería de datos y aplicaciones para limpieza y corrección de datos.

A continuación se definen algunas funciones de distancia y similitud utilizadas por el algoritmo de vecino más cercano.

Distancia de Levenshtein

Para conocer que tan semejantes son dos cadenas de texto y poder cuantificar el grado de semejanza entre ellas se deben comparar ambas cadenas y así definir la distancia de edición, esta distancia corresponde al costo mínimo de transformaciones para convertir una cadena en otra mediante operaciones de edición, estas operaciones podrían ser inserciones, borrados o la sustitución de un símbolo por otro. La distancia de edición fue ideada por el científico ruso Vladimir Levenshtein y evaluada por Wagner y Fischer a través de un algoritmo de programación dinámica. En (Levenshtein, 1965) y (Wagner, Fisher, 1974) se puede obtener más información con respecto a la distancia de edición.

En los ámbitos donde se utilizan cadenas de símbolos para representar los datos, como por ejemplo el reconocimiento de patrones, es usado este tipo de algoritmos, el cual considera que un alfabeto es un conjunto no vacío de símbolos y una cadena es una secuencia finita de dichos símbolos.

Distancia PPM

Como compresores de texto funcionan mediante la estimación del contenido de información de una cadena, si dos cadenas A y B son idénticos, comprimir A o comprimir A + B (concatenando las cadenas) deben producir una diferencia muy pequeña (idealmente, un bit adicional para indicar la presencia de la información redundante). Por otro lado, si A y B son muy diferentes, comprimiendo A y comprimiendo A + B debe producir diferencias dramáticas en la longitud (OpenRefine, 2014).

Simil

Es un algoritmo que compara dos cadenas de texto, inicia buscando la subcadena de textos comunes más larga, a partir de esa subcadena busca, tanto a la derecha como a la izquierda, el resto de subcadenas comunes forma recursiva hasta quedar con los caracteres que son totalmente diferentes. Suma los valores que representan las subcadenas comunes y este resultado lo divide entre el largo total de las subcadenas, obteniendo un valor que representa el porcentaje de similitud este va en un rango de 0 (completamente diferente) y 1 que sería la comparación de 2 registros idénticos. Según el ejemplo mostrado en (Nielsen, et al., 2010), la similitud en la comparación de las palabras ‘Pennsylvania’ y ‘Pencilvaneya’ mediante el algoritmo Simil es de 0.67 (67%). Este porcentaje es el resultado de las subcadenas de caracteres comunes ‘lvan’, ‘Pen’ y ‘a’, cuyo resultado obtenido de la suma de caracteres comunes 8 por

cada una de las cadenas comparadas para un total de 16 y la suma es dividida entre el largo total de los caracteres que conforman las 2 cadenas.

3.3.3 Algoritmos híbridos

Los algoritmos de vecino más cercanos son generalmente más efectivos que los algoritmos por colisión de llave, no obstante los algoritmos por colisión de llave son más eficientes, esto lleva al desarrollo de algoritmos que combinan ambas categorías para tratar de conseguir eficacia y eficiencia. Los algoritmos híbridos en algunos casos generan llaves para crear subconjuntos o generan grupos superpuestos sobre los que llevarán a cabo las comparaciones de vecino más cercano de esta forma evitan comparar todos los elementos contra todos los demás y así reducir la cantidad de comparaciones considerablemente.

Algoritmos de bloqueo

Esta técnica ha sido usada desde los años sesentas, existen seis métodos de bloqueo diferentes, si se desea profundizar más en el tema se recomienda la investigación de (Draisbach & Naumann, 2011). Los métodos de bloqueo utiliza llaves BKV's, para *particionar* y crear subconjuntos de registros de una o varias bases de datos. En el método de bloqueo tradicional, todos los registros que tienen el mismo BKV se insertan en un mismo bloque de forma única.

Luego de crear cada bloque el algoritmo realiza las búsquedas de pares iguales comparando un registro de cada bloque contra todos los demás que pertenecen al mismo bloque, por lo tanto es una técnica que combinada con otras se convierte en una opción muy eficiente. Si los BKV se han generado de forma correcta, se puede decir que los registros duplicados se encuentren en el mismo bloque. Esto indica que la decisión más importante para el buen desempeño de este algoritmo tiene que ver con los atributos utilizados para la generación de los BKV's, primero porque el tamaño de los bloques dependerá de la cantidad de registros con el

mismo BKV y esto tendrá impacto en el tiempo de ejecución, también porque la correcta clasificación de los registros en cada bloque dependerá de la calidad de los datos utilizados para generar la llave.

Tomando en cuenta las limitaciones y características de los algoritmos estudiados, se encuentra la oportunidad ya sea de adaptar alguno (s) de estos; pero, específicamente, para el castellano costarricense o crear uno que esté basado en la fusión de algunos de los algoritmos expuestos con las mejoras que se puedan realizar.

Canopy Clustering

Esta tecnica mejora en forma general los algoritmos de clasificación ya que reduce drásticamente el número de cálculos que se requieren, divide el proceso en 2 etapas, en la etapa inicial, genera grupos superpuestos llamados “Canopies”, que son subgrupos de elementos que tienen una distancia de similitud aproximada dado un umbral desde un punto central a partir de una métrica muy sencilla de calcular (métricas basadas en *tokens*) y decide que puntos estan definitivamente lejos. En la segunda etapa solo se calculan las distancias con la métrica de similitud más exigentes, estrictas, exactas y pesadas, para los puntos que pertenecen al mismo canopy. En la Figura 1 se presenta un ejemplo de la técnica Canopy Clustering.

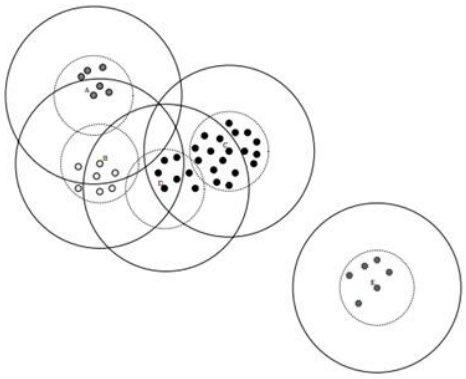


Figura 1: Ejemplo de Canopy Clustering. Fuente: Tomada de (McCallum, Nigamy, & H., 2000).

Los algoritmos canopy clustering, al ser una buena solución para clasificar grandes cantidades de datos, son utilizados por sistemas como Hadoop (sistema para almacenar, procesar y analizar grandes cantidades de datos el cual soporta aplicaciones distribuidas) ya que, en conjunto con el paradigma de programación Map Reduce, se convierte en una estrategia muy buena para el análisis de textos y procesamiento de imágenes de forma masiva (Garg, Trivedi, & B.B.Panchal, 2013).

4 Marco metodológico

4.1.1 Tipo y diseño de la investigación

4.1.2 Tipo de investigación

Una investigación pura parte de utilizar el método científico: se observa un fenómeno, se plantea una hipótesis, se hacen experimentos y de los resultados obtenidos en los experimentos se comprueba o refuta una hipótesis. Según (Málaga, Vera, & Oliveros, 2008) la investigación pura, llamada también básica o fundamental, tiene como objetivo mejorar el conocimiento per se, más que generar resultados o tecnologías que beneficien a la sociedad en el futuro inmediato. En el caso de la investigación Evaluativa como lo explica (Pelakais, Finol, Noel, & José, 2005) su objetivo es evaluar los resultados de uno o más programas, los cuales hayan sido, o están siendo, aplicados dentro de un contexto determinado.

Según estos conceptos, se establece que esta investigación es de tipo Pura y Evaluativa ya que lo que se pretende es generar conocimiento con respecto a las técnicas existentes que permiten solucionar problemas relacionados con la duplicación de datos y encontrar oportunidades de mejora, utilizando el método científico para el desarrollo de toda la investigación. Las técnicas serán sometidas a evaluación para comparar el nivel de eficiencia y eficacia.

4.1.3 Alcance de la investigación

Consultando a referentes de la investigación científica como (Hernández, Fernández, & Baptista, 2006), hacen una reflexión sobre la importancia del alcance, ya que de este dependerá la estrategia de la investigación. El alcance de una investigación es explicativa cuando se buscan encontrar las razones o causas que provocan cierto fenómeno y, así, atacar el problema en específico (p.116).

Tomando en cuenta que se realizaron diagnósticos, experimentos para encontrar las razones por las que se genera el problema planteado, y así proponer un diseño específico para tratar dicho problema, se define el alcance de esta investigación de tipo explicativa.

4.2 Enfoque de la investigación

Esta investigación utiliza un abordaje alternativo a la hora de definir el enfoque. Se parte de referencia de autores como (Chavarría, 2011) que insisten en la necesidad de reconocer que los enfoques cualitativos y cuantitativos no pueden existir de manera separada y que, más aún, no tiene sentido hablar de enfoques mixtos pues estos intentan mezclar de manera artificiosa cosas que nunca han estado desligadas. A continuación, se definen cada una de las dimensiones epistemológica, ontológica y axiológica de la investigación, sin hacer alusión explícita a un enfoque en particular.

La dimensión epistemológica, al tratarse del estudio de algoritmos existentes y contrastar resultados contra una solución nueva, implica que los investigadores tendrán una posición de observadores. Los investigadores prestarán atención a los detalles de eficacia, eficiencia, escalabilidad y facilidad de implementación. Por ello, las mediciones que se realicen deben efectuarse en condiciones similares para comprobar y documentar los resultados de los experimentos.

La dimensión ontológica, partiendo de algunos conceptos como el expresado por (Gruber, 1993): "lo que existe es exactamente aquello que puede ser representado". (Borst, 1997) lo define como “Una ontología es una especificación formal de una conceptualización compartida” (Citado por Arano, 2005). Studer y su grupo, en 1998, se encargaron de fusionar las definiciones de Gruber y Borst: “Una ontología es una especificación formal y explícita de una conceptualización compartida” (Studer, Benjamins V., & Fensel, 1998). Conceptualización se

refiere a un modelo abstracto de algún fenómeno en el mundo a través de la identificación de los conceptos relevantes de dicho fenómeno. Explícita significa que el tipo de conceptos y restricciones usados se definen explícitamente. Formal representa el hecho de que la ontología debería ser entendible por las máquinas. Compartida refleja la noción de que una ontología captura conocimiento consensual, esto es, que no es de un individuo, sino que es aceptado por el grupo".

El diccionario de la Real Academia de la Lengua Española define ontología como" Parte de la metafísica que trata del ser en general y de sus propiedades trascendentales". El primer campo en que se utilizó el término fue en la Filosofía; a partir de la década de los años noventa, el concepto de ontologías comienza a tener mayor importancia en las Ciencias de la Computación, entendiendo por ontología como un conjunto de términos básicos y relaciones entre sí. La representación del conocimiento requiere de mecanismos formales, para expresar con precisión la idea que se desea comunicar.

Este tipo de algoritmos ha sido ampliamente estudiado y es práctica común en las carreras de Informática recurrir a ellos para mostrar técnicas de programación. Por ello se considera que el estudio propuesto tiene una posibilidad clara de representación del conocimiento que se desea producir.

La representación ontológica de la metodología del algoritmo que se propone se puede apreciar en la Figura 2.

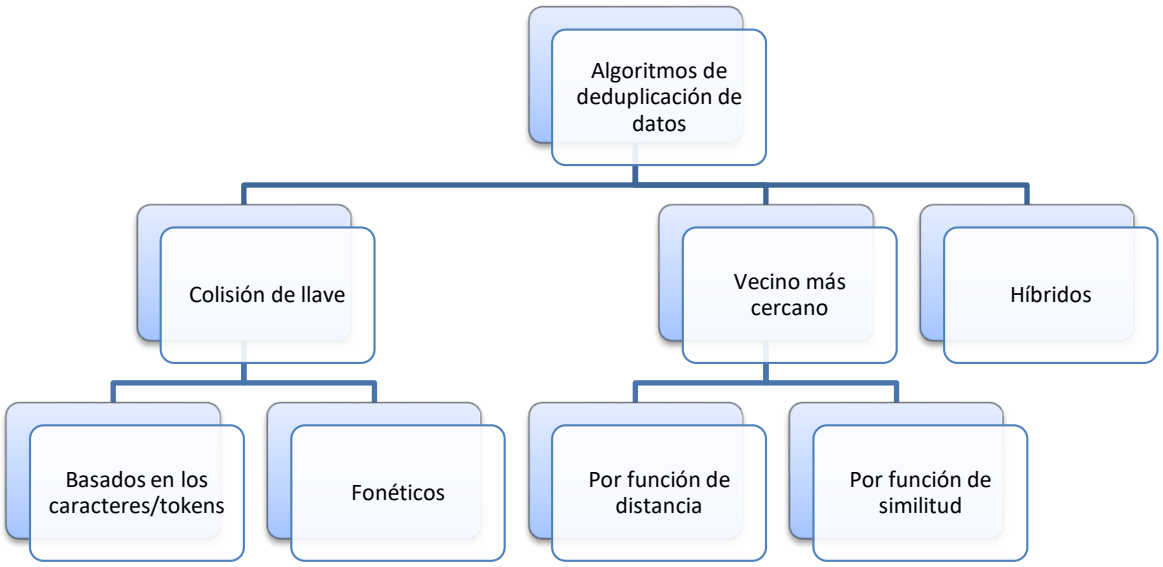


Figura 2: Ontología de Algoritmos de deduplicación basada en su metodología. Fuente: Elaboración propia.

El algoritmo que se propone busca acelerar el tiempo de ejecución de algoritmos de vecino más cercano y actuales algoritmos híbridos, y mantener una buena eficacia en los resultados.

La dimensión axiológica requiere especial tratamiento en cuanto a la facilidad de implementación del algoritmo. La eficacia, eficiencia y escalabilidad únicamente demandarán de los investigadores un esfuerzo por realizar mediciones exactas y documentarlas de manera adecuada. La facilidad de implementación, por otra parte, será evaluada de acuerdo a la escala de valores mostrada en la Tabla 2.

Tabla 2: Criterios de evaluación de la facilidad de implementación.

Rubro	**Valor**	**Opciones**		
Alternativas de implementación	20%	Opción única: 5%	Dos opciones: 10%	Más de dos opciones: 20%
Tipo de algoritmo	30%	Técnicas de IA: 10%	Técnicas convencionales con recursión: 15%	Código plano: 30%
Cantidad de líneas de código	35%	Más de 2000 líneas: 15%	De 1000 a 2000 líneas: 25%	Menos de 1000 líneas: 35%
Preparación de los datos	15%	3 tablas base o más: 5%	2 tablas base: 10%	1 tabla base: 15%

Entre mayor sea el porcentaje asignado, más fácil de implementar es el algoritmo. Se considerará que un 80% o más significan que el algoritmo es fácil de implementar, de un 50% a un 80% significa que es medianamente complejo de implementar; y menos de un 50% significa que es de implementación muy compleja.

4.3 Diseño de la Investigación

El diseño de una investigación es el plan o estrategia definido para responder a las preguntas de investigación; es decir, es lo que se debe realizar para alcanzar lo que se desea, el diseño de investigación puede ser experimental o no experimental. El término experimento, de forma general, refiere a “elegir o realizar una acción”; y, después, observar las consecuencias. De forma particular un experimento se lleva a cabo para analizar si una o más variables independientes afectan a una o más variables dependientes y porque lo hacen (Hernández, Fernández, & Baptista, 2006). En el caso de esta investigación, se aplica el diseño experimental, ya que se requiere, por medio de experimentos, comprobar y comparar los resultados tomando como medida las variables expuestas en la dimensión axiológica que son eficiencia, eficacia y facilidad de implementación entre el algoritmo desarrollado y los algoritmos investigados.

El diseño experimental debe cumplir con 3 requisitos, primero se debe manipular de forma intencional alguna(s) de las variables independientes. Segundo debe ser medible el efecto de una variable independiente sobre una dependiente y, por último, debe cumplir con control de validez internos del experimento.

El diseño transeccional se aplica en investigaciones donde los datos que se van a utilizar para el experimento se recopilan una única vez, en un solo momento.

Dado que los diferentes conjuntos de datos que serán utilizados para los experimentos a realizar durante esta investigación serán recolectados una única vez, los cuales se analizarán para comparar y medir las variables definidas, se determina que esta investigación cumple con un diseño transeccional experimental.

4.4 Técnicas e Instrumentos de medición

4.4.1 Técnicas

4.4.1.1 Experimentos

Una tarea experimental consiste en crear una situación controlada en la que se va a poner a prueba la hipótesis. Implica una mayor precisión de los términos y la posibilidad de replicarla exactamente (Rico, 1999).

La experimentación que se llevará a cabo en esta investigación es la siguiente:

4.4.1.1.1 Experimento 1

Escoger 22 individuos al azar, mayores de 18 años, ya sean estudiantes universitarios avanzados, profesionales entre 25-40 años, o profesionales mayores de 50 años, que constantemente realicen labores de digitación, hombres y mujeres. Se les solicitará mediante dictado digitar una lista de nombres de personas con un *teclado QWERTY* configurado para el español. Durante la escritura de los nombres dictados, los individuos no pueden realizar ninguna corrección, de manera que los investigadores puedan obtener una base de datos con los distintos factores y/o tipos de errores que se cometen a la hora de digitar y que producen valores duplicados, esto con el fin de analizar los parámetros que se deberán tomar en cuenta para el desarrollo de la propuesta. Para realizar el dictado, se utilizarán los nombres que se detallan en la Tabla 3. Lista de nombres personales.

4.4.1.1.2 Experimento 2

Probar el algoritmo desarrollado contra la base de datos de prueba y medir su eficacia.

4.4.1.1.3 Experimento 3

Ejecutar el algoritmo sobre un corpus donde se pueda medir la cantidad de coincidencias correctas, falsos positivos y tiempo de ejecución.

4.4.1.1.4 Experimento 4

Someter a pruebas y medición de eficiencia los algoritmos de deduplicación de datos existentes, previamente seleccionados, contra los 3 corpus anteriormente detallados para efectos de valoración.

4.4.1.2 Instrumentos

Tabla 3: Lista de nombres personales.

Nombre\Tipo Error	**Tipográfico Horizontal**	**Tipográfico Vertical**	**Fonético**	**Omisión de Letra**
María Gómez Jiménez				
Victor Fernández Salazar				
Lorena Guillén Salas				
Jorge Guadamuz Pérez				
Fabiola Quirós Castro				
Rolando Rivera Ortiz				
Josue Matarrita Porras				
Karla Cantillano Ureña				
Maritza Loaiciga Piñar				
Luis Soto Blanco				
Gladys María Ugalde Madriz				
Luis Roberto Céspedes Mora				
Santiago José Rodríguez Chavarría				
Francisco López Noguera				
Warren Castillo Gutiérrez				
Hector Felipe Quesada Carvajal				
Manfred Eduardo Salgado Romero				
William Mauricio Forester González				
Roger Manrique Otárola Esquivel				
Manuel Antonio Sibaja Méndez				
Total por Tipo Error				

Tabla 4: Lista de chequeo para evaluación de eficacia.

Número de registros: Número total de posibles comparaciones: Parámetros:		
	Valor obtenido	**Comentario**
Cantidad total de potenciales coincidencias		
Coincidencias correctas		
Cantidad de registros únicos en las potenciales coincidencias		
Coincidencias no detectadas		
Coincidencias filtradas		
Coincidencia filtradas correctas		
Coincidencias filtradas incorrectas (Falsos positivos)		
Coincidencias filtradas no detectadas.		

Tabla 5: Comparación de eficiencia.

Algoritmo	Duplicados encontrados	Tiempo de ejecución

Tabla 6: Evaluación de escalabilidad.

Cantidad de registros:		
Total de posibles comparaciones:		
Actividad	**Registros modificados o generados**	**Duración (s)**
Cálculo de coordenadas		
Indexación de columnas de ubicación		
Cálculo de todas las posibles coincidencias en determinado radio		
Cálculo coincidencias en determinado radio con cierta similitud o distancia determinada		
Total en segundos de coincidencias con cierta similitud o distancia determinada		

4.4.2 Análisis de datos

Analizar eficacia, eficiencia, facilidad de implementación y escalabilidad

4.4.2.1 Eficacia

Entendiéndose como "coincidencia" a cada par de valores en el corpus que cumplen con un umbral de distancia geométrica en cada dimensión, es decir:

$X = \{s,t\} / s \in C \ \wedge \ t \in C \ \wedge \ x_s - x_t <= radio_x \wedge y_s - y_t <= radio_y \ \wedge z_s - z_t <= radio_z \rightarrow \ s \approx t$

De manera que la evaluación de la eficacia de los algoritmos va a estar en función de los siguientes valores, los cuales serán detallados en la sección de análisis de resultados.

- Potenciales coincidencias
- Coincidencias correctas
- Cantidad de registros únicos en las potenciales coincidencias

- Coincidencias no detectadas
- Coincidencias filtradas
- Coincidencia filtradas correctas
- Coincidencias filtradas incorrectas (falsos positivos)
- Coincidencias filtradas no detectadas.

4.4.2.2 Eficiencia

La eficiencia del algoritmo será medida en función de las siguientes mediciones:

- Tiempo de ejecución
- Uso de memoria.
- Uso de procesador.
- Operaciones IO por segundo.

4.4.2.3 Facilidad de implementación

La facilidad de implementación será evaluada bajo los criterios definidos en la Tabla 2.

4.4.2.4 Escalabilidad

Será evaluada mediante utilizando como instrumento de la tabla 6.

5 Proceso de desarrollo

A continuación, se presenta el algoritmo llamado Geometric Distance for Strings (GDS), como propuesta de la presente investigación, una estrategia basada en el desarrollo de una función la cual, dada una cadena de texto S, devuelve las coordenadas (x_s, y_s, z_s), en un espacio tridimensional. El cálculo de dicha coordenada toma en cuenta los caracteres que contiene, secuencia de los caracteres y ubicación de estos en la cadena. Unido a esto, los investigadores al obtener los resultados del Experimento 1 deciden dar una mayor importancia dentro del cálculo del algoritmo a la distribución espacial de las letras en el teclado (véase Figura 3).

Tal y como se ha mencionado a lo largo de este documento, se hace uso del teclado QWERTY en español tanto para el algoritmo como para los ejemplos y experimentos. La importancia de la distribución espacial se debe a que de esta manera se permite mitigar errores tipográficos; de tal manera que si se aplica la función sobre cualquier otra cadena de texto R que tenga bastante similitud pero que no sea igual a S, y donde la distancia euclidiana entre ambas cadenas de texto sea consistente con su similitud, se puede decir que cuanto más similares sean las cadenas, más corta es su distancia geométrica.

Figura 3: Teclado QWERTY. Fuente: Adaptado de http://mkweb.bcgsc.ca/carpalx/images/qwerty.png.

La función de distancia de Levenshtein descrita en el apartado anterior, tiene como debilidad el hecho de que no considera la distribución espacial de las letras en el teclado, esto la hace un poco laxa (menos estricta). Un ejemplo de esto sería calcular la distancia de edición entre “MARIA” y “MARIO”: el cálculo presentará como resultado que la distancia sería de 1 y ambas se encuentran a la misma distancia que “MARIP”; sin embargo, cuando se considera la distribución de las letras en el teclado QWERTY se puede ver que resulta más probable que se haya escrito “MARIP”, cuando lo que se quiso escribir fue “MARIO”, esto dada la cercanía de la letra O con la letra P. Tal y como se muestra en la figura 4, tomando en cuenta la distribución espacial de las letras, la letra A se ubica bastante más lejos que la letra O; razón por la cual “MARIP” debería ser más similar y, por ende, más cercano a “MARIO” que “MARIA”.

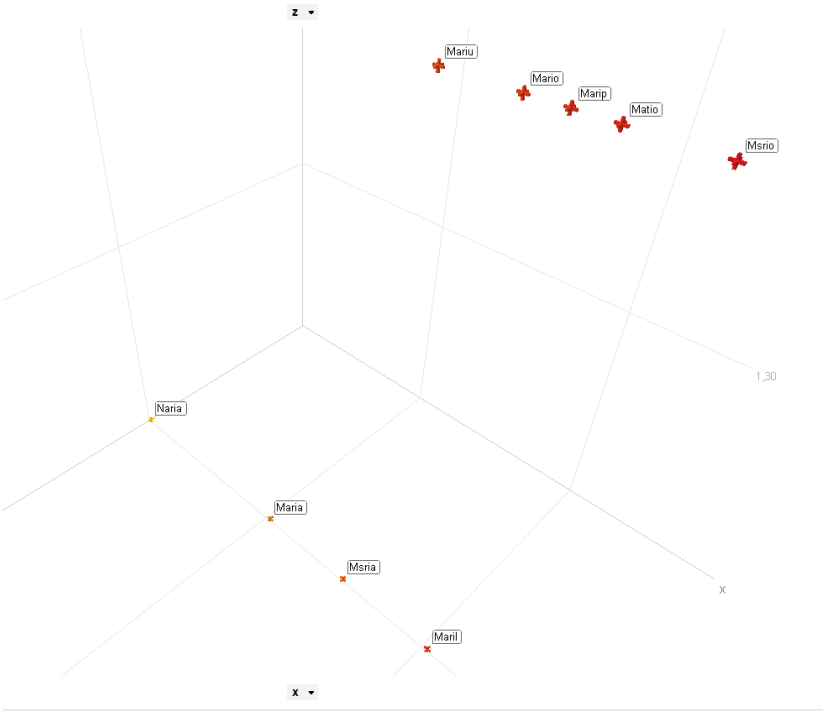


Figura 4: Ejemplo cercanía entre cadenas de texto similares según ubicación espacial. Fuente: Elaboración propia.

Se propone el planteamiento de la distribución espacial de las letras, como factor clave, de forma tal que esa colocación de cada letra aporte a la ubicación de una cadena de texto tras su ocurrencia. En la figura 5 se muestra la ubicación de las letras en un espacio dado.

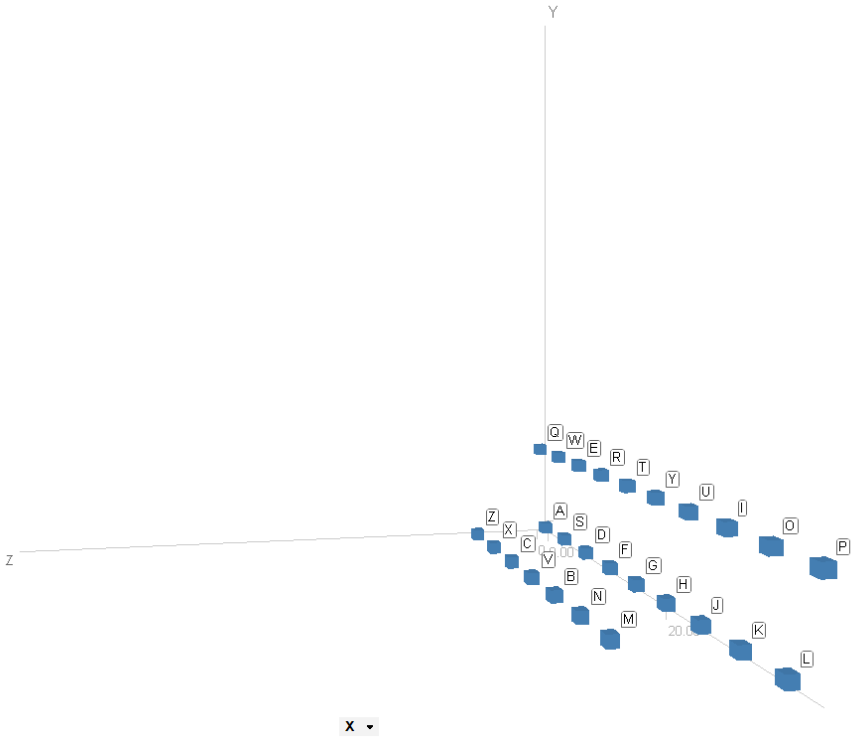


Figura 5: Ubicación espacial de las letras. Fuente: Elaboración propia.

Las 3 hileras de letras del teclado se han colocado en un espacio tridimensional con dimensión LxAxA, esto significa que cada par de letras contiguas horizontalmente tienen una distancia geométrica $\delta = L/9$, y $A = 1.5\delta$, esta distancia se calcula de esta manera para asignar más peso al error tipográfico horizontal que al error tipográfico vertical, esto se fundamenta ya que tomando como prueba los resultados obtenidos en el Experimento 1 el error con más coincidencias detectadas en la muestra es el error tipográfico horizontal; por lo tanto, la probabilidad de cometer este tipo de error a la hora de digitar es mucho mayor que los otros tipos de errores medidos, en el apartado de Análisis y Discusión de resultados se presentan y detallan los resultados obtenidos por medio de dicho experimento.

También, se ha agregado al sistema el desplazamiento horizontal que tienen las hileras de letras entre sí. Por ejemplo, si se toma la letra ‘Q’, que se encuentra alineada más a la izquierda en el teclado, esta se ubica en las coordenadas (0,A,0), entonces la letra ‘A’ horizontalmente será desplazada en el eje X, A/4 unidades a la derecha de la letra ‘Q’, Por lo que las coordenadas de la letra A sería (δ/4, 0, 0); con la misma lógica se desplaza horizontalmente la letra Z y sus coordenadas serían (3δ/4, 0, A). Se han creado las dimensiones del espacio tridimensional con las siguientes medidas 37x6x6. En la tabla 7, se presentan las coordenadas donde se ubica cada letra según las medidas en un teclado tipo QWERTY en español.

Tabla 7: Coordenadas de las letras en el espacio dado.

Letra	X	Y	Z
A	1	0	0
S	5	0	0
D	9	0	0
F	13	0	0
G	17	0	0
H	21	0	0
J	25	0	0
K	29	0	0
L	33	0	0
Ñ	37	0	0
Z	3	0	6
X	7	0	6
C	11	0	6
V	15	0	6
B	19	0	6
N	23	0	6
M	27	0	6
Q	0	6	0
W	4	6	0
E	8	6	0
R	12	6	0
T	16	6	0
Y	20	6	0
U	24	6	0
I	28	6	0
O	32	6	0
P	36	6	0

Para describir y dar mayor detalle a la forma como se consideraron las letras de la cadena y el orden de dicha cadena, se tiene asignada la ubicación de cada una de las letras en el espacio dado. El código de programación de la función se presenta en los anexos como el anexo #1.

fraccion es un método de la clase Coordenada3D, cuyo código fuente es el siguiente:

```
public void fraccion (Coordenada3D _coor, Double fraction) /* Calcula un punto W que se encuentra a la fracción del segmento entre el punto actual y el punto que recibe como parámetro */  {
            this.x = this.x  +(_coor.x - this.x )*fraction ;
            this.y =  this.y  +(_coor.y - this.y )*fraction ;
            this.z = this.z  +(_coor.z - this.z )*fraction ;
            }
```

5.1 Descripción del Algoritmo.

La ubicación espacial se ha calculado basándose en los siguientes factores de cada cadena: caracteres presentes, secuencia de los caracteres y ubicación de los caracteres en la cadena, considerando los caracteres a la izquierda como los más significativos y los caracteres a la derecha como los menos significativos, por lo que cuánto más a la izquierda el caracter aportará más a su ubicación en el espacio, este último factor es muy significativo cuando los nombres se encuentran en formato <APELLIDOS><NOMBRES> que es el caso de los corpus utilizados, donde la combinación de apellidos es menos recurrente que los nombres y por tanto, más significativos; y los nombres, a veces, suelen ingresarse de manera incompleta, por ejemplo: “Castro Rodriguez Maria de los Ang” o “Solís Matamoros Luis C.”. La combinación de los tres factores producirá resultados menos laxos que cualquier otro subconjunto de factores.

El algoritmo primero toma la cadena a ubicar y le aplica transformaciones para remover acentos con excepción de la “ñ”, signos de puntuación y control (códigos de caracteres que no son visibles) y se convierten a mayúsculas todos los caracteres que conforman la cadena. Dentro del mismo proceso de transformación, el algoritmo no reordena los *tokens*; sin embargo, este es

un parámetro que se pueda activar cuando se conoce que en las cadenas donde se va aplicar el algoritmo se tiene un orden distinto de los *tokens*, por ejemplo si en el conjunto aparecen nombres con el siguiente orden de los *tokens* “Maria Fernanda Villegas Nuñez”, y otros con orden distinto como el siguiente “Villegas Nuñez Maria Fernanda”, sin esa opción parametrizable el algoritmo no detectará estos duplicados porque quedarán distantemente ubicados ya que, para la ubicación espacial, los caracteres a la izquierda tienen mayor valor significativo que los de la derecha, a menos que se aplique el reordenamiento de los *tokens* y entonces quedarán con distancia 0.

Como segundo paso el algoritmo recorre la cadena caracter por caracter en orden inverso, obtiene la ubicación inicial en el espacio tomando las coordenadas donde se encuentra la primera letra de la cadena invertida. Si la cadena tiene más caracteres, se calcula una fracción (parámetro entre 0 y 1) de la distancia entre la ubicación inicial y la distancia donde se encuentra ubicada la siguiente letra. La nueva ubicación obtenida será el punto de partida para calcular la siguiente ubicación, que se realiza de forma exacta al paso anterior, este proceso se realiza de forma cíclica hasta llegar a la última letra de la cadena, devolviendo las coordenadas de la última ubicación generada como la ubicación espacial de la cadena de texto.

En la Figura 6, se describe cómo funciona el algoritmo mediante un ejemplo. Utilizando la cadena de caracteres que forman la palabra “Rosa”, el algoritmo iniciaría el recorrido con la letra “A” que sería la letra primera en orden inverso y calcularía la ubicación inicial como la ubicación en el espacio dado donde se encuentra esta letra, la siguiente ubicación se calcula partiendo de esta ubicación inicial más la proporción dada al algoritmo como parámetro: por ejemplo 1/2 de la distancia hacia donde se encuentra la siguiente letra que sería la letra “S”, obteniendo esta nueva ubicación el algoritmo haría el mismo proceso hasta llegar a la letra “R”,

con lo que se obtienen las coordenadas donde se ubica la palabra “Rosa” en un espacio determinado.

La complejidad computacional de este método para el cálculo de las coordenadas es O(n), siendo n el número de caracteres que posee la cadena, es un procedimiento muy sencillo, pero usa un modelo efectivo para la tolerancia de errores tipográficos, detalles que se mostrarán en la sección de análisis de resultados.

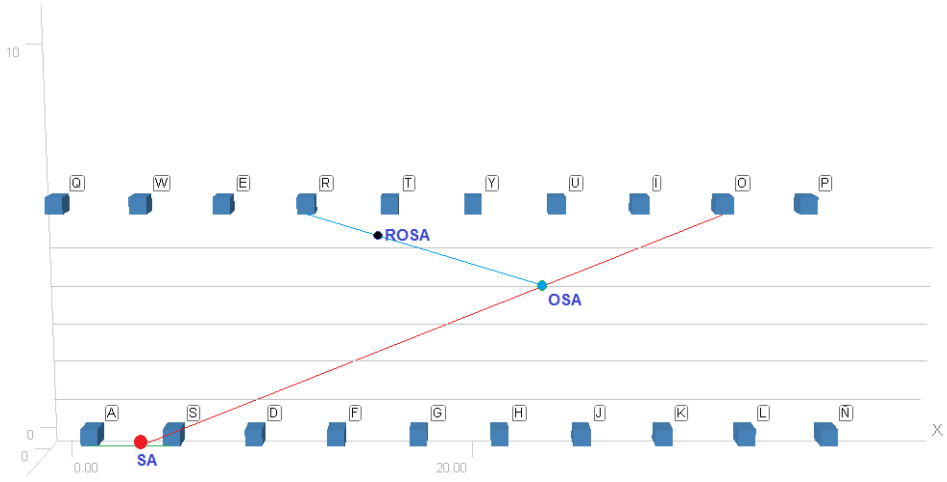


Figura 6: Ejemplo Ubicación espacial de una palabra. Fuente: Elaboración propia.

5.2 Limitaciones y debilidades del algoritmo

Según se explicó en la sección 5.1, el orden de los caracteres en la cadena es muy relevante para el cálculo de las coordenadas, y los caracteres a la izquierda aportan mayor peso en la ubicación que los caracteres a la derecha, lo que hace el procedimiento sensible a las diferencias entre cadenas en los primeros caracteres, ya que podrían quedar muy alejadas y fuera del radio asignado. Este problema podría presentarse más cuando se utilice la opción de reordenamiento de tokens, pues se pueden encontrar dos cadenas que siendo muy similares una

posea un token más que la otra, si este token debido a la letra inicial afecta el orden de inicio de toda la cadena esto podría provocar que se ubicaran en zonas muy alejadas y no se reconocieran como duplicados; lo mismo puede ocurrir en aquellos casos donde las cadenas en los datos fuentes no se encuentren ubicadas de acuerdo a los valores más significativos y completos a los menos significativos. De la misma manera para cadenas muy largas el algoritmo pierde efectividad en la ubicación esto se debe a que los caracteres más a la derecha aportan un valor muy poco significativo y por tanto las cadenas de longitud mayor a 60 caracteres podrían tener un aporte casi nulo de los últimos caracteres.

6 Análisis y discusión de resultados

En este apartado se presentan los resultados de la ejecución de los experimentos llevados a cabo tanto para el análisis antes y después del desarrollo del algoritmo GDS.

6.1 Análisis y resultados antes del desarrollo del algoritmo

Para este análisis, se llevó a cabo el experimento 1, de las 22 muestras obtenidas, se compararon letra por letra los nombres dictados a la población seleccionada contra los datos suministrados para el desarrollo del ejercicio. Luego, de realizada la comparación se unifican los datos para tener los resultados generales. En la Tabla 8 se muestran dichos resultados.

Tabla 8: Resultado Experimento1 Lista de nombres personales.

Nombre\Tipo Error	**Tipográfico Horizontal**	**Tipográfico Vertical**	**Fonético**	**Omisión de Letra**
María Gómez Jiménez	2	0	2	2
Victor Fernández Salazar	1	1	1	10
Lorena Guillén Salas	2	1	1	5
Jorge Guadamuz Pérez	4	1	1	0
Fabiola Quirós Castro	1	1	1	0
Rolando Rivera Ortiz	1	1	2	3
Josué Matarrita Porras	3	0	0	4
Karla Cantillano Ureña	3	1	8	2
Maritza Loaiciga Piñar	4	4	4	9
Luis Soto Blanco	2	0	0	3
Gladys María Ugalde Madriz	5	0	8	0
Luis Roberto Céspedes Mora	2	2	1	3
Santiago José Rodríguez Chavarría	3	1	1	2
Francisco López Noguera	4	2	0	2
Warren Castillo Gutiérrez	5	0	2	2
Hector Felipe Quesada Carvajal	3	4	2	4
Manfred Eduardo Salgado Romero	3	2	0	1
William Mauricio Forester González	7	3	0	1
Roger Manrique Otárola Esquivel	6	2	1	0
Manuel Antonio Sibaja Méndez	2	2	1	1
Total por Tipo Error	**64**	**28**	**36**	**55**

Tal y como se observa en la Tabla 8, los errores más comunes encontrados son los tipográficos horizontales. Dado que parte de lo que buscaban los investigadores antes de desarrollar la propuesta era conocer con qué frecuencia se cometen más errores tipográficos horizontales versus errores tipográficos verticales a la hora de digitar utilizando un teclado QWERTY. Se divide este tipo de error en 2 categorías. Luego de aplicar el experimento se encuentra que el error tipográfico horizontal como el más recurrente, esto muestra que es muy probable que, por la cercanía de las teclas y el movimiento tan pequeño que se debe realizar al digitar teclas ubicadas en la misma línea horizontal, se convierte en un error muy común, de los 22 individuos; solamente 2 de ellos no cometieron este tipo de error. Además, se analiza que de los 64 errores tipográficos horizontales encontrados, solamente en 6 casos se digitó alguna tecla de la misma línea pero, con más de una tecla de distancia entre ellas. Los errores tipográficos verticales son los casos menos encontrados en dicho experimento.

Los errores por omisiones de letras o fonéticos fueron contabilizados pero la propuesta no se inclinará a tratar este tipo de errores. Partiendo de esto nace un especial interés sobre los errores tipográficos y la ubicación espacial de las letras en el teclado QWERTY en español como factores de mayor relevancia.

6.2 Análisis y resultados del algoritmo GDS

Para efectos de evaluar el algoritmo y las comparaciones con algoritmos semejantes se ejecutó el algoritmo GDS sobre diferentes corpus. El primer corpus, con 2000 registros de personas, representa el conjunto de datos que fue manipulado para obtener pares de cadenas de texto muy similares pero no iguales, con igual número de cédula, el largo de dichas cadenas también se generó de forma intencional para poder medir el comportamiento del algoritmo con cadenas de diferente longitud con un largo máximo de sesenta caracteres. Al ser este un ambiente

controlado, se obtuvieron de forma anticipada por medio de consultas SQL la cantidad exacta de cadenas de texto similares candidatas a ser datos duplicados. Para efectos de identificación, será llamado Corpus1. En esta primera parte se muestran los resultados sobre el Corpus1. La ubicación de los nombres espacialmente se pueden observar en la Figura 7.

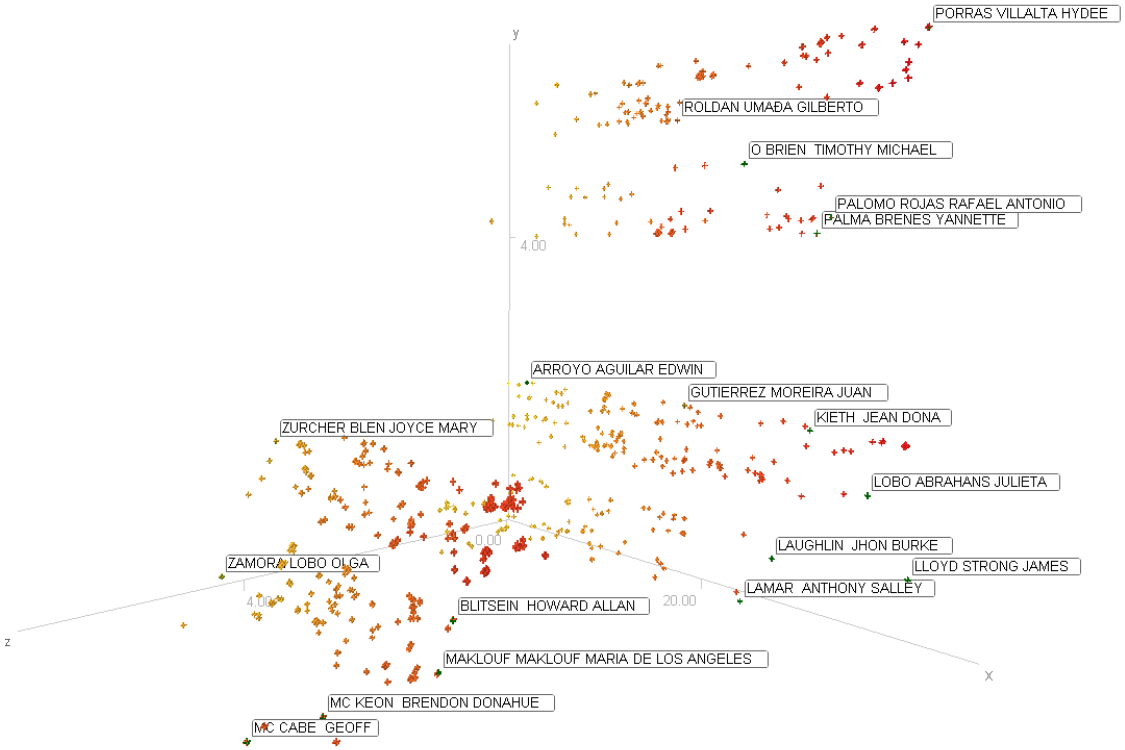


Figura 7: Ubicación espacial Corpus1. Fuente: Elaboración propia.

En la Figura 8 se presenta un acercamiento de algunos registros específicos para hacer notar lo cercano que quedan, espacialmente, los valores que son similares.

Figura 8: Cadenas de caracteres similares en el espacio. Elaboración propia.

La estrategia propuesta mediante el algoritmo GDS agrega los valores X, Y y Z al Corpus utilizado, incluso permite crear índices sobre ellos para la ejecución de consultas optimizadas en la búsqueda de posibles registros duplicados o búsquedas difusas, delimitando los registros a comparar por posición espacial las cadenas de texto; es decir, cuando se quieran calcular las distancias y similitudes entre los registros, no hace falta comparar todos los registros contra todos sino, que la comparación se realiza solamente contra los vecinos de cada elemento que se encuentra en cierto espacio alrededor. Mediante el script SQL #1 se realiza dicha comparación.

```
SELECT
c1.id,
c1.[persona],
c1.[numeroIdentificacion],
c2.[id],
c2.[persona],
c2.[numeroIdentificacion],
SQRT(POWER(c1.x - c2.x,2)+POWER(c1.y - c2.y,2)+POWER(c1.z -    c2.z,2)) AS
distancia_geometrica,[Corpus1].[dbo].[fnSimil](c1.persona, c2.persona) AS
similitudFROM
[Corpus1].[dbo].[ CorpusLocated] c1
LEFT OUTER JOIN [Corpus1].[dbo].[CorpusLocated] c2
```

```
ON c1.id < c2.id
AND c2.x between c1.x - 0.00003 AND c1.x + 0.00003
AND c2.y between c1.y - 0.00003 AND c1.y + 0.00003
AND c2.z between c1.z - 0.00003 AND c1.z + 0.00003
WHERE c2.persona IS NOT NULL
ORDER BY distancia_geometrica
```

A continuación, en la Tabla 9, se presenta una muestra de los resultados de la ejecución de la consulta sobre Corpus1, para efectos de un mejor análisis se muestran solamente los 10 nombres más cercanos (con distancia diferente de 0):

Tabla 9: Los 10 nombres más cercanos.

Nombre1	**Nombre2**	**distancia geométrica**	**similitud**
CHACON MARIN MARIA DELIA CC. MARIA ADE	CHACON MARIN MARIA DELIA CC.MARIA ADEL	9.95E-14	0.974358974
MONGE MENA MARIA LIDIA C.C.LILLIAN	MONGE MENA MARIA LIDIA CC LILLIAM	2.98E-13	0.927536232
ESQUIVEL SABORIO FRANCISCO C.C. FANKLIN	ESQUIVEL SABORIO FRANCISCO CC FRANKLIN	5.93E-13	0.962025316
TIMM SCOTT CHRISTOPHER CHARLE	TIMM SCOTT CHRISTOPHER CHARLES	1.30E-12	0.96875
STEINVORTH JIMENEZ CARL WALTER	STEINVORTH JIMENEZ CARL WALTER C/C CARLOS	1.61E-12	0.849315068
RODRIGUEZ JIMENEZ CARLOS ENRI	RODRIGUEZ JIMENEZ CARLOS ENRIQUE	3.23E-12	0.952380952
GONZALEZ VARGAS JORGE ALBERTO	GONZALEZ VARGAS JORGE ALBERTO RAMON	4.69E-12	0.909090909
JIMENEZ AGUILAR MARIA EUGENIA	JIMENEZ AGUILAR MARIA EUGENIA C.C. CASTRO	4.78E-12	0.833333333
CHAVES CASTRO RAMON FRANCISCO	CHAVES CASTRO RAMON FRANCISCO CC CARLOS	4.86E-12	0.857142857
CHAVARRIA ARLEY MARIA CECILIA	CHAVARRIA ARLEY MARIA CECILIA CC MARIA HA	5.22E-12	0.833333333

Tal y como se puede notar, los valores de la distancia geométrica están en notación científica y representan valores muy pequeños; por ejemplo: el primer valor 9.95E-14, corresponde a una distancia de 0.0000000000000995

En la Tabla 10, se muestran los últimos 10 resultados de la consulta con mayores distancias:

Tabla 10: Los 10 nombres más lejanos para el umbral dado.

Nombre1	Nombre2	distancia geométrica	similitud
RODRIGUEZ FONSECA JACINTO	RODRIGUEZ HERNANDEZ EZEQUIEL	3.21E-05	0.545454545
RODRIGUEZ FONSECA JACINTO VICTOR	RODRIGUEZ HERNANDEZ EZEQUIEL	3.21E-05	0.483870968
RODRIGUEZ FONSECA JACINTO VICTOR	RODRIGUEZ HERNANDEZ EZEQUIEL.	3.21E-05	0.476190476
RODRIGUEZ FONSECA VICTOR	RODRIGUEZ HERNANDEZ EZEQUIEL.	3.21E-05	0.545454545
RODRIGUEZ FONSECA VICTOR	RODRIGUEZ HERNANDEZ EZEQUIEL	3.21E-05	0.555555556
PEREZ CHAVES MANUEL FRANCISCO	PEREZ CHAVES JOSE FRANCISCO	3.38E-05	0.862068966
PEREZ CHAVEZ MANUEL FRANCISCO	PEREZ CHAVEZ JOSE FRANCISCO	3.38E-05	0.862068966
ROCHA ROCHA SAIDY	ROCHA ROCHA SEIDY	3.47E-05	0.944444444
CASTRO MONGE MARIA ROSA	CASTRO MONGE ROSA MARIA	4.18E-05	0.791666667
VARGAS LOPEZ MARLENE	VARGAS LOPEZ ROSA MARLENE	4.29E-05	0.893617021
ACUÑA RIVERA CARMEN RUDY	ACUÑA RIVERA RUDY	4.71E-05	0.837209302

Distancia de los valores candidatos a ser duplicados reales

Durante los experimentos, fue muy importante realizar el análisis de las distancias determinadas para cada uno de los pares de valores que verdaderamente correspondían a valores duplicados, esto para efectos de identificar los valores duplicados que fueron determinados como los registros que comparten el mismo número de identificación o cédula. Para llevar a cabo el análisis de los pares se ha implementado el script SQL #2, en este caso aplicado sobre el Corpus1:

```
SELECT
      C1.[ROW],
     C1.[CEDULA],
     C1.[NOMBRE],
abs(C1.[x] - C2.x) AS diffx,
     abs(C1.[y] - C2.y) AS diffy,
abs(C1.[z] - C2.z) AS diffz,
SQRT(POWER(C1.x - C2.x,2)+POWER(C1.y - C2.y,2)+POWER(C1.z - C2.z,2)) AS
distancia_geometrica,
```

```
    C2.[ROW],
    C2.[CEDULA],
    C2.[NOMBRE]
FROM
        [Corpus1].[dbo].[CorpusLocated] C1
INNER JOIN [Corpus1].[dbo].[CorpusLocated] c2
ON c1.ROW < C2.ROW AND C1.CEDULA = C2.CEDULA
ORDERBY distancia_geometrica
```

La Figura 9 muestra, mediante un histograma, la distribución de las distancias geométricas de los valores que previamente se habían determinado como los valores que corresponden a duplicados.

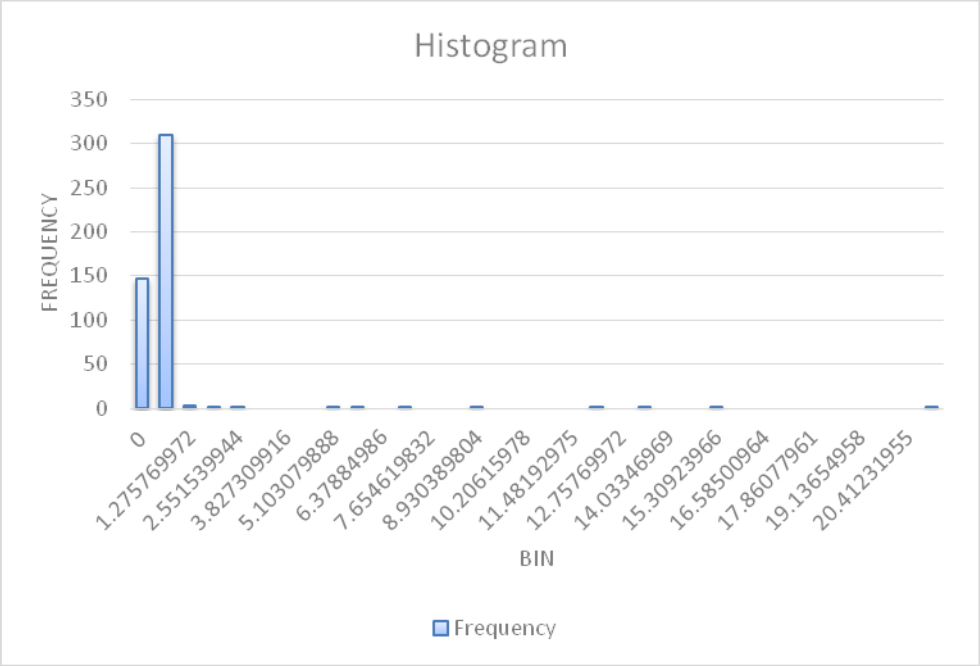


Figura 9: Distribución de distancias geométricas. Fuente: Elaboración propia.

En el histograma se puede apreciar algunos pares muy separados, que corresponden a valores duplicados lo que significa que esa información corresponde a la misma persona pero sus nombres han sido digitados con un orden diferente de los *tokens* o en otros casos del todo

representan *tokens* distintos. La Tabla 11 muestra un ejemplo de 10 pares de nombres que reflejan los casos comentados anteriormente:

Tabla 11: Pares muy separados de valores duplicados.

Nombre 1	**Nombre 2**	**distancia geométrica**
NASO ROCCO RICHARD	ROCCO RICHARD NASSO	22.87546095
GARZA RAMIREZ RICARD O	RODRIGUEZ RAMIREZ RI CARDO	22.93657975
IMMETHUN LAWRENCE PAUL	LAWRENCE PAUL	23.32669379
COOK PRENTICE DEAN	DEAN COOK PRENTICE	24.2921678
LORIA MADRIGAL RODOLFO	MADRIGAL LORIA RODOLFO CC RODOLFO LORIA	24.68130113
LORIA MADRIGAL RODOLFO	MADRIGAL LORIA RODOLFO	24.68130114
NASSO ROCCO RICHARD	ROCCO RICHARD NASSO	24.71873883
LAWRENCE EDWARD WORKMAN	WORKMAN LAWRENCE EDWARD	25.68730764
DINO NASTASI	NASTASIS GINO	25.83213534
BOLAÐOS CANTON LUIS ENRIQUE	VARGAS CANTON LUIS ENRIQUE	26.21473599

La distancia tan separada por causa del orden de los *tokens* se puede resolver mediante la aplicación de un reordenamiento de los mismos antes de calcular las coordenadas, pero para efectos del algoritmo planteado en la presente investigación, el orden de todos los caracteres es muy relevante para su ubicación espacial; por lo tanto, depende de la naturaleza de las cadenas de texto el elegir si se desea aplicar esa transformación o no, como se describió previamente esta opción es parametrizable a la hora de aplicar el algoritmo.

En las figuras 10 y 11 se presenta un acercamiento al espacio entre 0 y 1 del histograma de las distancias geométricas entre valores duplicados verdaderos:

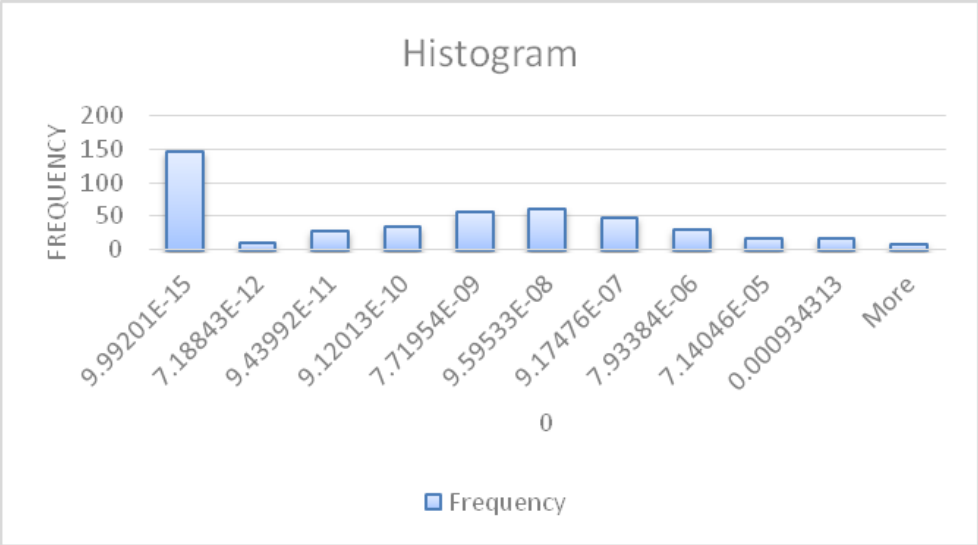


Figura 10: Distancias geométricas de valores duplicados reales entre 0 y 1. Fuente: Elaboración propia

.

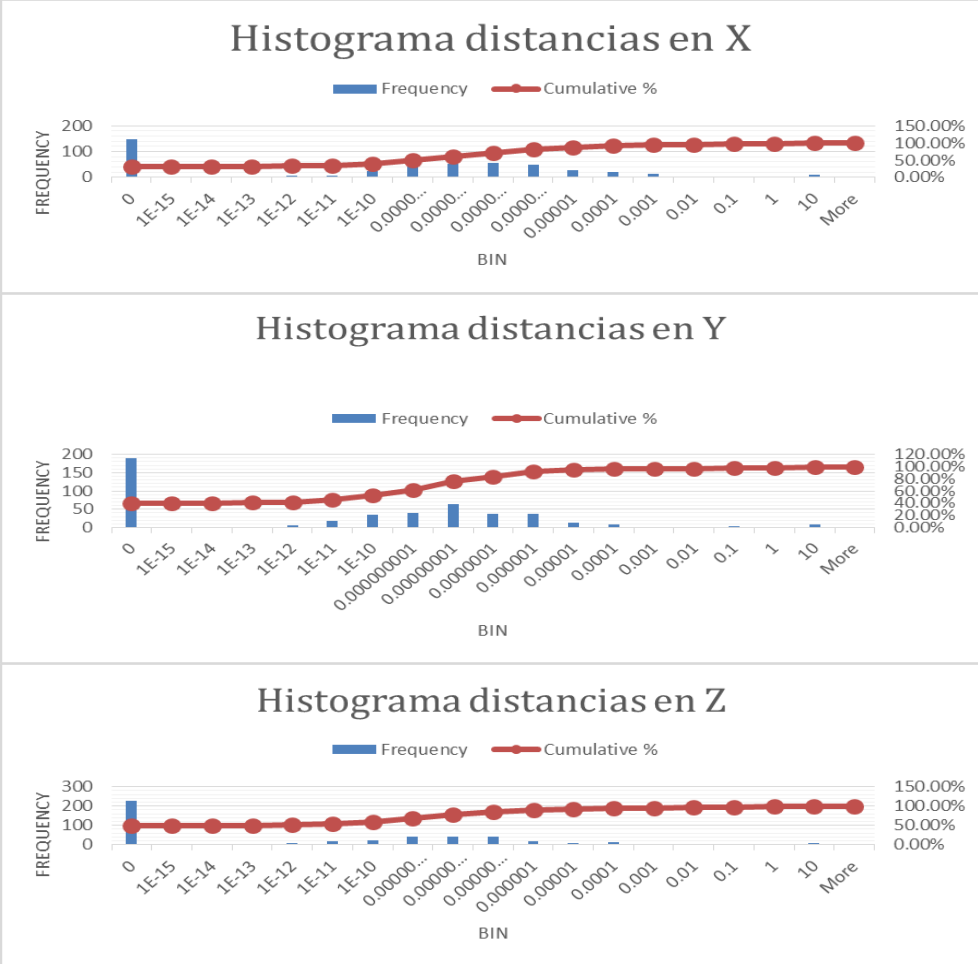


Figura 11: Histograma de diferencias en los ejes X,Y,Z de valores duplicados reales. Fuente: Elaboración propia.

Con los resultados de la comparación de las distancias geométricas se pudo identificar que, cuanto más largas eran las cadenas, más nanométricas se volvía la distancia entre ellas, esto siempre y cuando sean similares pero no idénticas, lo que lleva a los investigadores a definir los umbrales de comparación en función de la longitud de las cadenas; de lo contrario, se podría ser muy laxo con las cadenas largas o muy estricto con las cadenas cortas.

La Figura 12 corresponde a un gráfico de dispersión entre la longitud promedio de las cadenas comparadas versus la distancia geométrica en escala logarítmica.

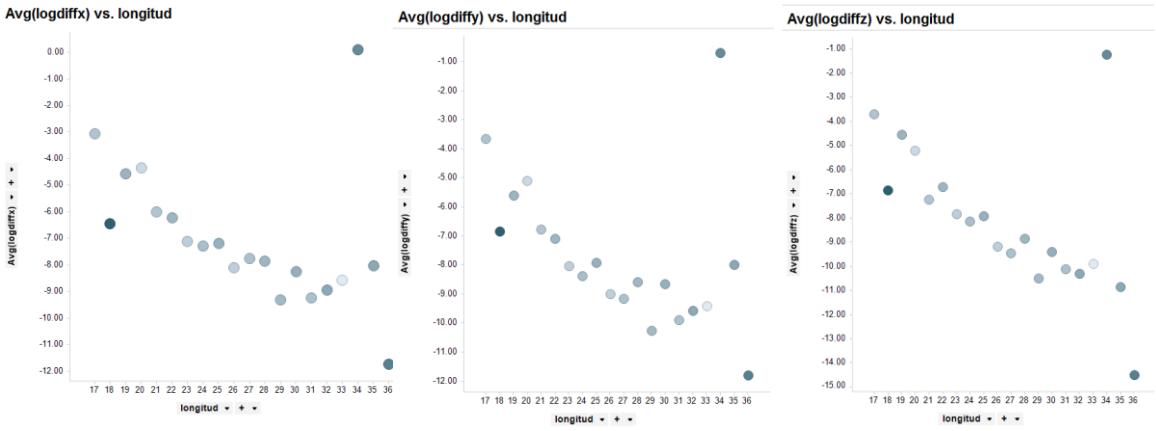


Figura 12: Distancias geométricas (escala logarítmica) vs la longitud promedio de cadenas duplicadas reales. Fuente: Elaboración propia.

Con el análisis de las distancias geométricas en escala logarítmica en los 3 ejes con respecto a la longitud de las cadenas de texto comparadas, se determina que lo más eficaz y eficiente no es calcular “radios” fijos óptimos sino funciones que calculen el umbral óptimo en función de la longitud de las cadenas, se pueden calcular las siguientes ecuaciones de las líneas de tendencia:

$$d_x \leq 10^{-0.123l - 1.9413}$$

$$d_y \leq 10^{-0.186l - 3.5179}$$

$$d_z \leq 10^{-0.2869l - 1.0592}$$

Los coeficientes de estas ecuaciones han sido los valores óptimos para el Corpus2 utilizado pero si se quiere estos valores pueden ser modificados dependiendo del nivel de similitud o que tan difusos sean los resultados que se requieren.

Estas ecuaciones forman parte de la programación para la generación de clústeres de valores duplicados o muy similares porque definen los límites del sector en el espacio tridimensional (un *ortoedro*) donde se seleccionan las cadenas con las que se hará la comparación más intensa, es decir, el cálculo de la similitud que terminará de filtrar los valores a considerar duplicados.

La Figura 13 fue muy importante para determinar el valor que debía tener al parámetro que representa el valor definido como porcentaje de similitud, este valor es clave para condicionar la generación de los clústeres ya que estos se van a crear tomando en cuenta los registros con un grado de similitud el cual puede ser <, >, = al valor dado a la función Símil. Este parámetro, al igual que todos los demás, necesitan ajustarse específicamente según el corpus que se utilice, por tal motivo los valores definidos para el análisis de los resultados presentados en esta sección son específicos para el Corpus2, este corpus corresponde a una base de datos real con un poco más de 15500 registros. Con la ayuda del siguiente histograma se analizaron las distancias determinadas por la ubicación espacial del algoritmo, y así se encontró el valor apropiado para los parámetros requeridos.

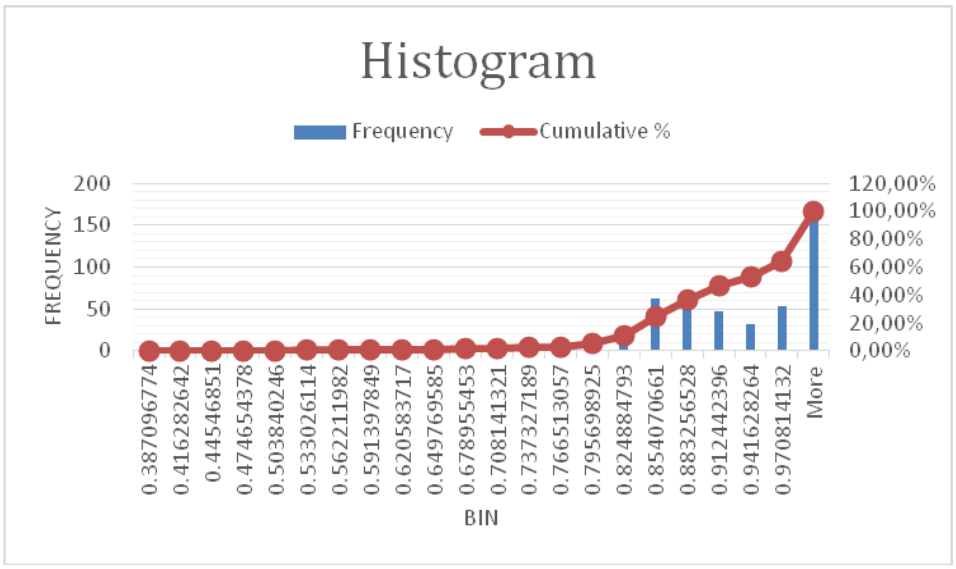


Figura 13: Similitud de valores duplicados reales entre 0 y 1. Fuente: Elaboración Propia.

Analizando la información que muestran las figuras anteriores y después de varias pruebas, se elige un porcentaje de similitud mayor al 88% para los posibles pares de duplicados.

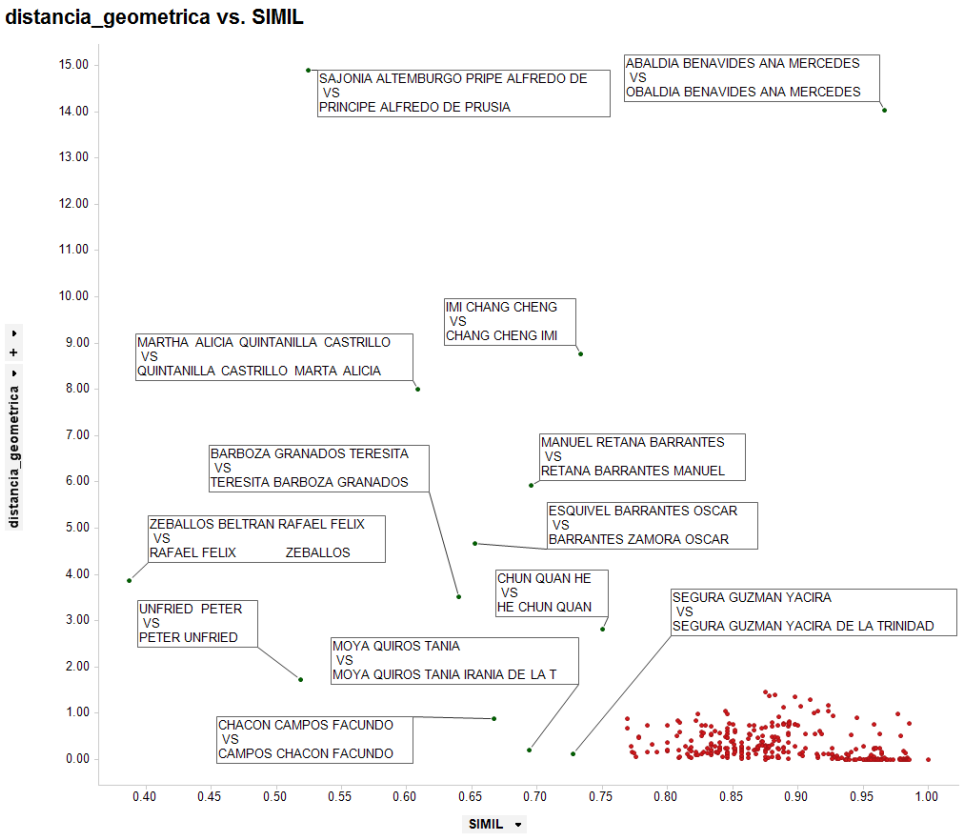


Figura 14: Gráfico de dispersión distancia geométrica vs valor de similitud para duplicados reales. Fuente: Elaboración propia.

El gráfico de la figura 14 es un gráfico de dispersión donde se colocan los valores duplicados reales del Corpus2, mediante este se compara la distancia geométrica correspondiente vs su valor de similitud, se puede observar que la mayoría de los valores se encuentra donde se espera encontrarlos, muy cercanos al 1 en similitud y muy cercanos al 0 en distancia geométrica, los demás valores que se encuentran alejados son aquellos duplicados que tienen diferente orden de *tokens,* para lo cual ambos algoritmos son sensibles.

6.2.1 Resultados análisis de la eficacia

Para llevar a cabo el Experimento 2, mediante el cual se evalúa la eficacia, se utilizó como instrumento de medición la lista de chequeo para la evaluación de la eficacia, presentada en el apartado 4 en la Sección de Instrumentos.

A continuación, en la Tabla 12 se presenta el instrumento descrito con los resultados obtenidos durante el análisis de resultados del algoritmo GDS.

Tabla 12: Evaluación de la eficacia GDS.

Número de registros:**15521**
Número total de posibles comparaciones: **120.44×10^6**
Parámetros:

- fracción = 0.5
- radioX = radioY = radioZ = 0.001

	Valor obtenido	**Comentario**
Cantidad total de potenciales coincidencias	23,133	Las más de 120 millones de posibles comparaciones para la detección de duplicados será podada a 23,133 comparaciones
Coincidencias correctas	413	Valor esperado : 470 Las 23,133 comparaciones seleccionadas detectarán 413 de 470 valores duplicados reales, es decir 87.9% de los duplicados reales.
Cantidad de registros únicos en las potenciales coincidencias	7999	La cantidad de registros a utilizar de 15521 para encontrar el 87.9% de los valores duplicados reales
Coincidencias no detectadas	57	La cantidad de pares de duplicados reales que fueron podados por los radios seleccionados.
Coincidencias filtradas	834	Después de filtrar potenciales coincidencias con una función de similitud (SIMIL) mayor al 88% quedan 834 posibles pares de duplicados

Coincidencia filtradas correctas	270	De los 834 pares filtrados solo 270 si corresponden
Coincidencias filtradas incorrectas (Falsos positivos)	564	De los 834 pares filtrados 564 no son duplicados reales (se determina por la cédula)
Coincidencias filtradas no detectadas.	143	De las 413 coincidencias correctas en el umbral, se descartaron 143 tras filtrarlos con una similitud superior al 88%

A continuación, se muestra los pasos ejecutados para obtener los resultados presentados en el instrumento anteriormente mostrado, para el algoritmo GDS:

Generación de potenciales coincidencias

Después de haber calculado las ubicaciones en el espacio tridimensional de cada cadena de texto para el Corpus2 se calculan los registros de pares de potenciales valores duplicados o potenciales coincidencias; es decir, los pares de valores que tienen una proximidad menor o igual al radio dado, estos pares son los que finalmente se pueden someter a un proceso más pesado de comparación como alguna función de similitud o de distancia para afinar la detección de duplicados.

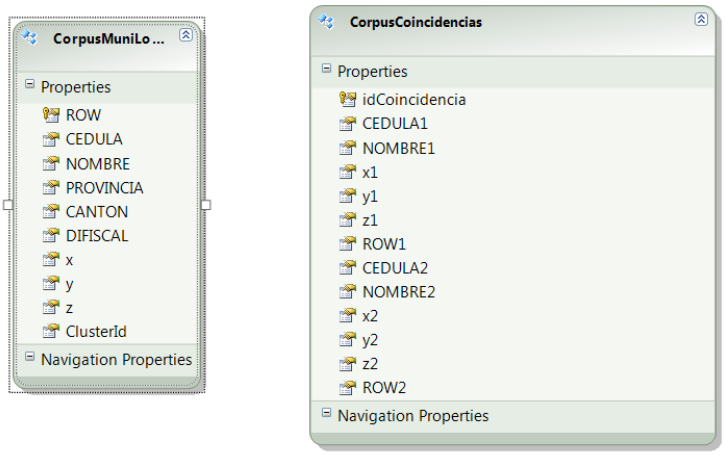


Figura 15: Tablas bases de datos para generación de coincidencias. Fuente: Elaboración propia.

En la Figura 15 se muestran el diagrama de las tablas utilizadas para almacenar los datos generados durante la ejecución del algoritmo.

El código de programación para la generación de una tabla auxiliar con los registros de potenciales coincidencias es el siguiente:

```
foreach (CorpusMuniLocated currentRecord in corpusIndex.Skip(preclusteredCurrPageNumber *
preclusteredPageSize).Take(preclusteredPageSize).ToList())
                {
double radioX = 0.001;
double radioY = 0.001;
double radioZ = 0.001;
List<CorpusMuniLocated> vecinos =
                    tfgBL.getRecordsNear(currentRecord, radioX, radioY, radioZ);
foreach (CorpusMuniLocated matchingRecord in vecinos)
                    {
if (
Math.Abs(currentRecord.x - matchingRecord.x) < radioX
&&Math.Abs(currentRecord.y - matchingRecord.y) < radioY
&&Math.Abs(currentRecord.z - matchingRecord.z) < radioZ
/* &&   SimilCLR.clsSimil.similDotNet(currentRecord.NOMBRE,
                            matchingRecord.NOMBRE) >= 0.70 */
)
                        {
                          tfgBL.AddCoincidencia(currentRecord, matchingRecord);
                        }
                    }
                }
```

El equivalente en código SQL, sin el uso de la tabla auxiliar de coincidencias, sería la consulta SQL #3:

```
SELECT c1.[ROW]
,c1.[CEDULA]
,c1.[NOMBRE]
,c2.[ROW]
,c2.[CEDULA]
,c2.[NOMBRE]
FROM [Corpus].[dbo].[CorpusMuniLocated] c1 leftouterjoin
[Corpus].[dbo].[CorpusMuniLocated] c2 on c1.ROW< c2.ROW
where c2.x between c1.x - 0.001  and c1.x + 0.001
and c2.y between c1.y - 0.001  and c1.y + 0.001
and c2.z between c1.z - 0.001  and c1.z + 0.001
```

El código anterior para cada uno de los valores del corpus 2 evalúa cuáles de sus vecinos están a una distancia en el umbral y registra cada par en la tabla llamada Corpus Coincidencias.

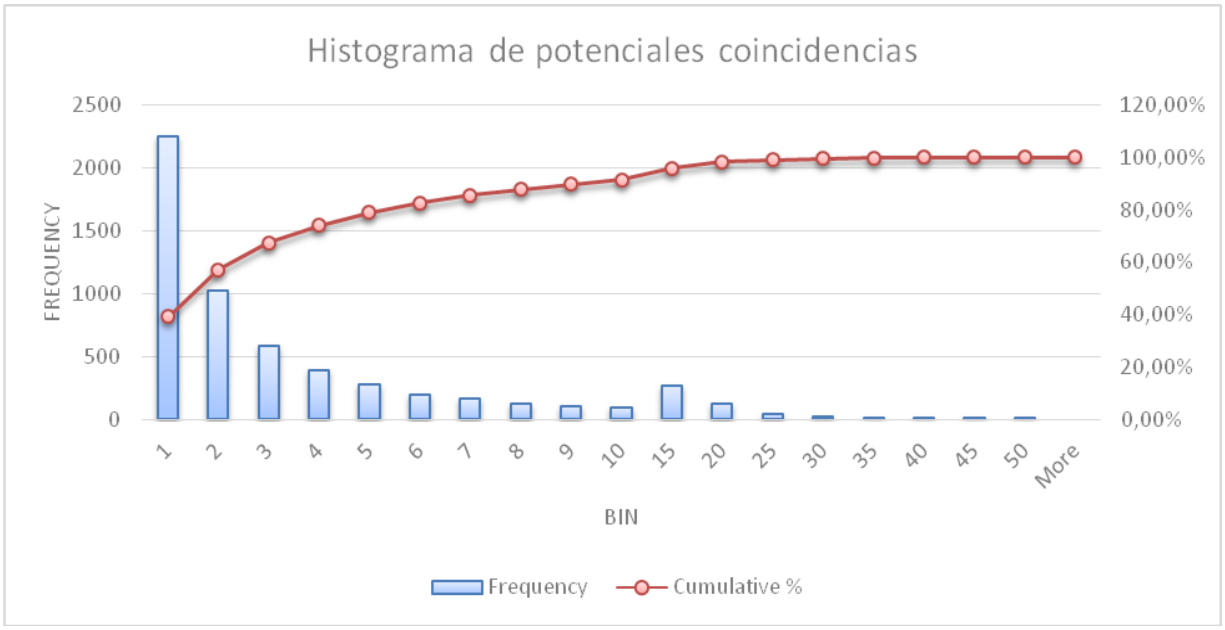


Figura 16: Histograma de la cantidad de registros con N potenciales coincidencias. Fuente: Elaboración propia.

La figura 16 representa cuántos registros encontraron N potenciales coincidencias, satisfactoriamente se obtiene que el 80% de los registros tienen 4 o menos potenciales comparaciones por ejecutar para encontrar sus posibles duplicados, y que para unos pocos valores requiere más de 30 comparaciones hasta un máximo de 48 comparaciones.

Coincidencias correctas: Se considera como coincidencia correcta, cada par de valores que el proceso determinó como duplicados y que, efectivamente, corresponden a duplicados reales porque coinciden sus números de cédula. El resultado de las coincidencias correctas fue obtenido mediante la consulta SQL #4:

```
SELECT
[idCoincidencia],
[ROW1],
[CEDULA1],
[NOMBRE1],
french2.dbo.fnSimil(NOMBRE1,NOMBRE2)AS simil,
ABS([x1] - [x2])as difx,
ABS([y1] - [y2])as dify,
ABS([z1] - [z2])as difz,
[CEDULA2],
```

```
[NOMBRE2],
[ROW2]
FROM
	[Corpus].[dbo].[CorpusCoincidencias]
WHERE CEDULA1 = CEDULA2
```

Cantidad de registros únicos en las potenciales coincidencias: Estos registros son los elementos únicos que fueron comparados dentro del total de registros de potenciales coincidencias.

Coincidencias no detectadas: Se consideren coincidencias no detectadas los pares de duplicados no detectados a pesar de que sus números de identificación son iguales. A continuación, se presenta la consulta SQL#5, mediante la cual se obtienen la lista de coincidencias no detectadas:

```
SELECT
[idCoincidencia],
[ROW1],
[CEDULA1],
[NOMBRE1],
french2.dbo.fnSimil(NOMBRE1,NOMBRE2)AS simil,
ABS([x1] - [x2])as difx,
ABS([y1] - [y2])as dify,
ABS([z1] - [z2])as difz,
[CEDULA2],
[NOMBRE2],
[ROW2]
FROM
	[Corpus].[dbo].[CorpusCoincidencias]
WHERE french2.dbo.fnSimil(NOMBRE1,NOMBRE2)<= 0.88
AND CEDULA1 = CEDULA2
```

Coincidencias filtradas: Se considera una coincidencia filtrada a cada una de las potenciales coincidencias que cumpla con el porcentaje de similitud aplicado por medio de la función de similitud SIMIL, sobre el conjunto total de posibles coincidencias, se pueden obtener con la consulta SQL #7.

```
SELECT c1.[ROW]
,c1.[CEDULA]
,c1.[NOMBRE]
,c2.[ROW]
,c2.[CEDULA]
,c2.[NOMBRE]
FROM [Corpus].[dbo].[CorpusMuniLocated] c1 leftouterjoin
[Corpus].[dbo].[CorpusMuniLocated] c2 on c1.ROW< c2.ROW
where c2.x between c1.x - 0.001  and c1.x + 0.001
and c2.y between c1.y - 0.001  and c1.y + 0.001
and c2.z between c1.z - 0.001  and c1.z + 0.001
and Corpus.dbo.fnSimil(c1.NOMBRE,c2.NOMBRE)> 0.88
```

Coincidencias filtradas correctas: Se considera una coincidencia filtrada correcta a cada una de las coincidencias correctas que cumpla con el porcentaje de similitud aplicado por medio de la función de similitud SIMIL, sobre el conjunto total de posibles coincidencias.

Coincidencias filtradas incorrectas (Falsos Positivos): Se considera un falso positivo cada par de valores, los cuales han sido detectados por el algoritmo como pares de registros duplicados. Estos pares se encuentran dentro de los umbrales definidos pero no son pares de valores idénticos y se conoce que en la "vida real" no corresponden a la misma entidad ya que poseen números de cédula distintos. A continuación, se presenta la consulta SQL #8 mediante la cual se obtienen la lista de falsos positivos:

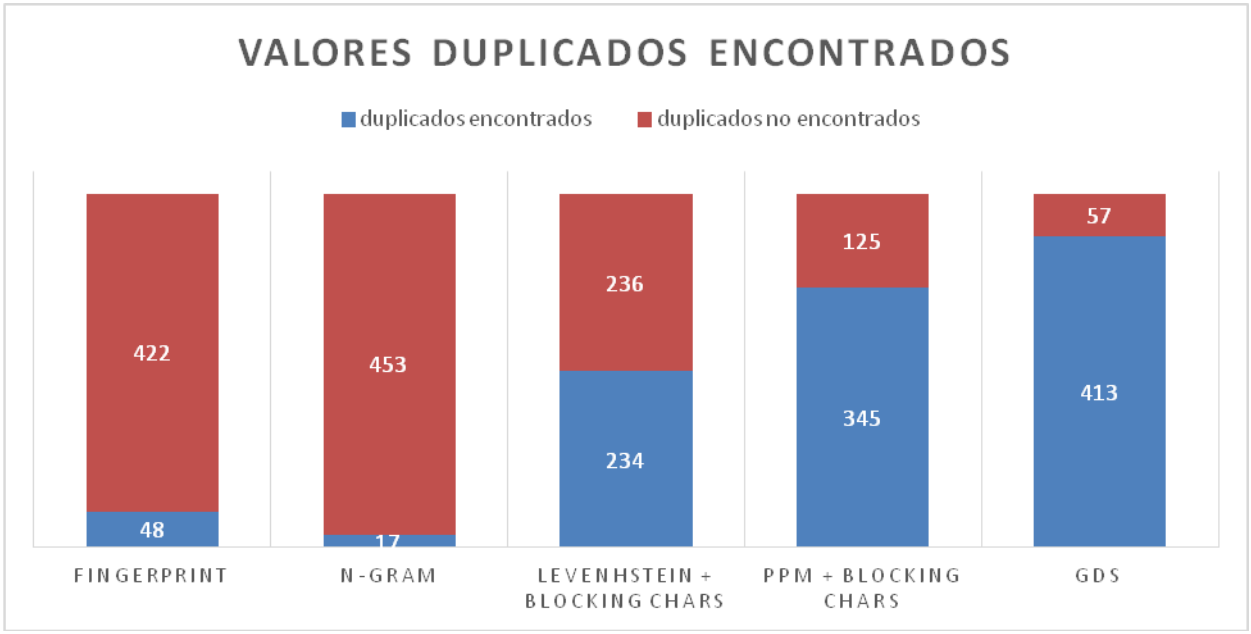
```
SELECT
[idCoincidencia],
[ROW1],
[CEDULA1],
[NOMBRE1],
french2.dbo.fnSimil(NOMBRE1,NOMBRE2)AS simil,
ABS([x1] - [x2])as difx,
ABS([y1] - [y2])as dify,
ABS([z1] - [z2])as difz,
[CEDULA2],
[NOMBRE2],
[ROW2]
FROM
       [Corpus].[dbo].[CorpusCoincidencias]
WHERE french2.dbo.fnSimil(NOMBRE1,NOMBRE2)> 0.88
AND CEDULA1 <> CEDULA2
```

Coincidencias filtradas no detectadas: Se considera una coincidencia filtrada no detectada a los pares de duplicados no detectados después de aplicada la función de similitud SIMIL con el porcentaje definido sobre todos los registros de posibles coincidencias correctas.

6.2.2 Resultados análisis de la eficiencia

Para las siguientes pruebas se ha utilizado la programación del *software* google-refine 2.5 el cual tal y como se describió anteriormente trae implementaciones de los algoritmos de colisión de llave como Fingerprint y N-gram, y algoritmos híbridos que corresponden a implementaciones de Levenshtein y PPM combinados con el algoritmo de bloques de caracteres como técnica para podado de comparaciones.

Las pruebas se ejecutaron sobre el corpus 2 donde de antemano se conoce que tiene 470 valores duplicados reales. En la figura 17 se muestran los resultados obtenidos.



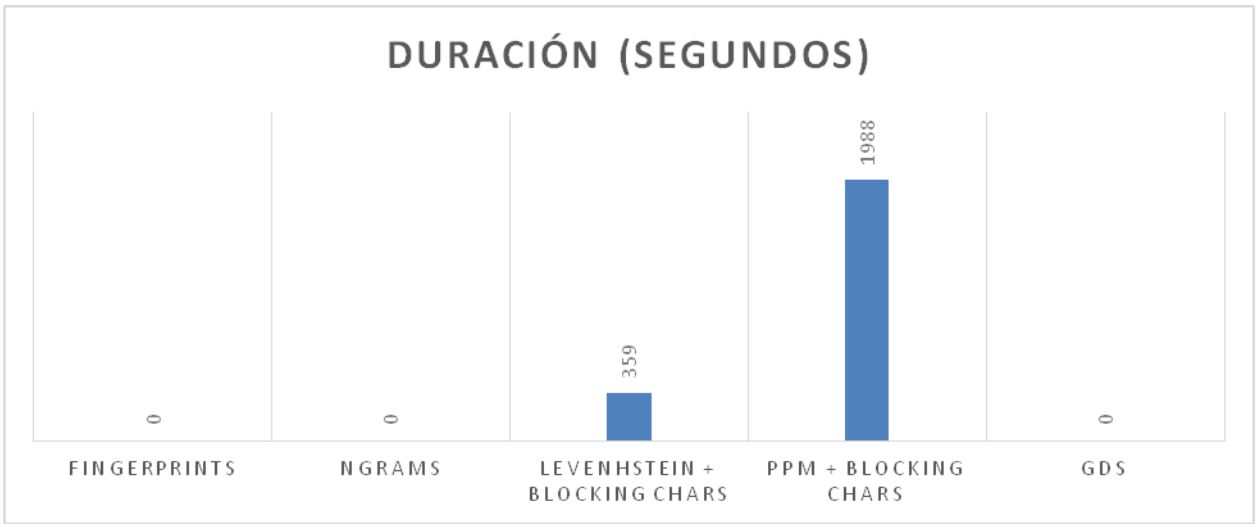


Figura 177: Análisis de eficiencia con respecto a la eficacia: Elaboración Propia.

Optimización de las consultas

Para efectos de la presente investigación el motor de base de datos utilizado es SQL Server 2014 Express. Un aspecto importante para mejorar el rendimiento de las consultas fue analizar la necesidad de utilizar índices. Para esto se tomó en cuenta principalmente la información referente a los campos que almacenan las coordenadas, tal y como se había mencionado anteriormente en las consultas se realizaron las comparaciones sobre los elementos que se encuentren en un espacio definido y no contra todos los elementos que se encuentren en el corpus por lo tanto se creó un índice nonclustered que incluye los 3 campos (x, y, z). Se obtuvo el plan de ejecución de la consulta y se confirma que el optimizador elige el índice creado y el tiempo de respuesta mejoró de forma exitosa. En la sección de anexos se puede encontrar el código generado para la creación de dicho índice y el plan de ejecución del Script SQL#1 utilizando el Corpus con más de un millón de registros.

6.2.3 Resultados y análisis de la Facilidad de implementación

Para medir el grado de facilidad de implementación del Algoritmo GDS, se tomaron en cuenta las opciones y rangos de valores establecidos en la tabla 2 Criterios de evaluación de la facilidad de implementación. Dado los resultados obtenidos mostrados en la tabla 13, se considera que dicho algoritmo es fácil de implementar ya que el valor obtenido es mayor al 80% establecido como rango a partir del cual se considera un algoritmo fácil de implementar.

Tabla 13. Evaluación de la facilidad de implementación.

Rubro	Valor Obtenido	Opciones
Alternativas de implementación	15%	Más de dos opciones: 15%
Tipo de algoritmo	25%	Código plano: 25%
Cantidad de líneas de código	35%	Menos de 1000 líneas: 35%
Preparación de los datos	15%	2 tablas base: 10%
Total de % Obtenido	**95%**	

En la opción de las alternativas de implementación se le asignó el máximo valor definido, ya que el algoritmo GDS se puede implementar en diferentes lenguajes de programación, así como también en diferentes motores de bases de datos. El tipo de algoritmo es de líneas de código plano, no utiliza recursión, ni técnicas de Inteligencia Artificial. La cantidad de líneas de código no supera las 500 por lo tanto cumple con la condición de ser un código de menos de 1000 líneas razón por la cual se le asigna también el máximo porcentaje definido. Para la preparación de los datos y la generación de los mismos se utilizan 2 tablas, la tabla origen de datos donde después de aplicado el algoritmo se agregan 3 columnas con las distancias calculadas para cada registro, también se crea una segunda tabla CorpusCoincidencias, donde se almacenan los registros que son potenciales coincidencias, al utilizar 2 tablas se ubica dentro del valor medio definido para

medir la preparación de los datos, el total de % obtenido es de un 95% de facilidad de implementación.

6.2.4 Resultados análisis de la Escalabilidad

Para analizar la escalabilidad del algoritmo se ha probado sobre un corpus que posee más de un millón de registros, para el cual buscar valores duplicados puede llevar a más de seiscientos mil millones de comparaciones, sin embargo aplicando el algoritmo GDS y creando el índice sobre las columnas de coordenadas, se pueden obtener casi 60 000 pares de valores con similitud mayor a un 88% en un tiempo aproximado a 2 minutos.

Tabla 144. Evaluación de la escalabilidad del algoritmo GDS + SIMIL.

Cantidad de registros:	**1,169,416**	
Total de posibles comparaciones:	**683,766,305,820**	
Actividad	Registros modificados o generados	Duración (s)
Cálculo de coordenadas	**1,169,416**	**50**
Indexación de columnas de ubicación	**1,169,416**	**10**
Cálculo de todas las posibles coincidencias en el radio 0.001	4,884,852	**91**
Cálculo coincidencias en el radio 0.001 con similitud > 88%	**57765**	**57**
Total en segundos de coincidencias con similitud > 88%		**117 = 50+10+57**

7 Conclusiones

Se detallan a continuación las principales conclusiones que se derivan de la presente investigación:

Se identificó que los errores más comunes que pueden generar datos duplicados utilizando un teclado QWERTY latinoamericano como instrumento para ingresar datos, son los errores tipográficos representados por el 50% de los errores que se generaron en el experimento. Por lo tanto, ubicar en el espacio tridimensional sobre el cual el algoritmo GDS calcula las distancias según el orden que tienen las letras en el teclado QWERTY si fue muy relevante para mitigar este tipo de errores.

Se seleccionaron cuatro algoritmos para la detección de valores duplicados los criterios para la selección fueron en primer lugar los algoritmos que más se utilizaron en las diferentes investigaciones elegidas en el estado de la cuestión, estos algoritmos son Levenshtein distance y N-gram. En segundo lugar se seleccionaron los algoritmos Fingerprint y PPM los cuales son utilizados por la aplicación para limpieza de datos OpenRefine.

Se formuló como proceso de poda el algoritmo GDS, el cual calcula coordenadas de cada cadena de texto en un espacio tridimensional delimitado, con dichas coordenadas se puede elegir a los vecinos más cercanos de cada cadena dentro de un radio determinado y realizar comparaciones solo contra los elementos que pertenecen a la zona definida. En una de las pruebas realizadas con los parámetros utilizados se logró podar hasta el 99.98% de todas las posibles comparaciones y solo se descartaron el 12.1% de los valores duplicados reales.

Se aplicó sobre cinco algoritmos tanto GDS como cuatro existentes sobre uno de los corpus de tamaño medio y en todos los casos se ejecutó el proceso finalizando de forma exitosa.

Asimismo se aplicó exitosamente el algoritmo GDS sobre 3 corpus distintos y de esta forma se respalda que el comportamiento del algoritmo fue constante.

Las mediciones de los resultados obtenidos con la aplicación de los algoritmos seleccionados y GDS fueron satisfactorias en cuanto a eficiencia, eficacia, escalabilidad y facilidad de implementación. GDS fue el algoritmo que se ejecutó en el menor tiempo con una duración menor a un segundo y fue el que detectó la mayor cantidad de valores duplicados 413 de los 470 valores existentes en el corpus seleccionado.

Se valora la eficacia y eficiencia del algoritmo GDS de la siguiente manera: una vez culminadas todas las mediciones y análisis realizados y se evidencian los resultados obtenidos en los experimentos los cuales demuestran que ambos factores a valorar han sido superados de forma considerable por GDS con respecto a los algoritmos seleccionados para la comparación. La creación de índices sobre las coordenadas generadas por GDS, optimizan la ejecución de las búsquedas de posibles registros duplicados, esta es una de las mayores ventajas por las cuales el algoritmo es tan eficiente.

Se comprueba la hipótesis planteada en esta investigación y se demuestra que tanto la eficiencia como la eficacia han sido superadas de forma considerable por GDS con respecto a los algoritmos seleccionados para la comparación, esto dada la ubicación espacial de las letras según el orden en el que se ubican en un teclado QWERTY latinoamericano. Esta estrategia ha sido satisfactoria para corregir errores tipográficos, estos según se demuestra en esta investigación representan un alto porcentaje de las causas mediante las cuales se generan errores a la hora de ingresar datos mediante esta configuración de teclado.

8 Consideraciones Finales

En la presente investigación se analizó el uso del algoritmo GDS como apoyo a la detección de valores duplicados, sin embargo, una muy importante aplicación puede ser la optimización de búsquedas difusas, donde dada una tabla que posea al menos una columna varchar sobre la cual se puedan efectuar búsquedas difusas, se mantengan precalculadas las coordenadas de la ubicación de esos registros para que ante una eventual búsqueda se calculen las coordenadas de la cadena a localizar y se filtre los registros en la zona determinada, así se puede lograr descartar numéricamente a muchas otras posibles valores o comparaciones.

El algoritmo GDS se puede utilizar en el desarrollo de una nueva especie de índice para búsqueda difusas sobre campos de texto, de manera que cuando se crea el índice en una tabla sobre una columna específica, se crearía una estructura de datos ordenada separada que consideraría las coordenadas del valor del registro y mantendría un puntero al registro correspondiente, por lo tanto, se podría usar búsqueda difusa sobre los campos de una tabla que tuviera N atributos varchar sin tener que crear en la misma tabla las 3xN columnas que representan las coordenadas de cada atributo.

9 Líneas futuras de investigación

Este trabajo de investigación genera nuevas inquietudes, interrogantes e ideas que abren oportunidad a nuevas líneas de investigación. A continuación, se presentan algunas líneas de investigación que pueden ser de interés para trabajos futuros.

Con respecto a la relación de GDS con el teclado QWERTY se puede realizar un análisis de factibilidad sobre diferentes configuraciones de teclados, incluso otros tipos de teclados tales como Dvorak. Así se pueden estudiar las distintas configuraciones y analizar si los errores tipográficos son tan recurrentes en estas otras configuraciones como se encontró en el teclado QWERTY y encontrar oportunidades de mejora para que GDS no solamente sea una solución efectiva y eficiente para un teclado en específico.

Otra oportunidad es incluir dentro de GDS el teclado numérico y tratar campos donde se almacenen números como texto; de esta forma, no solamente estaría ligado a la detección de nombres de personas duplicados sino también a campos como, por ejemplo, cédulas de identidad, de esta manera se podría ampliar el alcance del algoritmo a tratar errores tipográficos en el teclado numérico.

Podría ser importante desarrollar un algoritmo de ubicación espacial que considere el factor fonético para de esta manera mitigar este tipo de error de generación de datos duplicados.

11 Referencias

Amón, I., Moreno, F., & Echeverri, J. (2012). Algoritmo Fonético para detección de cadenas de texto duplicadas en el idioma Español. *Revista Ingenierías Universidad de Medellín*. Obtenido de http://webapps.udem.edu.co/RevistaIngenierias/pdf/v11n20/Art%EDculo%2010.pdf

Aspell. (2014). Obtenido de Lawrence Philips' Metaphone Algorithm: http://aspell.net/metaphone/

Chavarría, C. (2011). La Dicotomía cuantitativo/cualitativo: falsos dilemas en investigación social . *Actualidades en Psicología*, 1-35. Obtenido de http://www.revistas.ucr.ac.cr/index.php/actualidades/article/view/70/62

Christen, P. (2006). *A Comparison of Personal Name Matching: Techniques and Practical Issues.* IEEE CONFERENCE PUBLICATIONS.

Christen, P. (2012). A Survey of Indexing Techniques for Scalable. *Knowledge and Data Engineering, IEEE Transactions on*.

Citado por Arano, S. (2005). *Los tesauros y las ontologías en la Biblioteconomía y la Documentación.* Barcelona: Universitat Pompeu Fabra. Obtenido de http://ddd.uab.cat/pub/artpub/2003/88756/hipertext_a2003n1a11/tesauros.html

Draisbach, U., & Naumann, F. (2011). A Generalization of Blocking and Windowing. *Data and Knowledge Engineering (ICDKE), 2011 International Conference on.* IEEE CONFERENCE PUBLICATIONS.

Eckerson, W. (2002). *Data Quality and the Bottom Line.*The Data WareHousing Institute. Recuperado el 18 de agosto del 2014, de http://download.101com.com/pub/tdwi/Files/DQReport.pdf

Gallardo, H. (1999). *Elementos de Investigación Académica.* San José: Editorial Universidad Estatal a Distancia.

García, J. F. (2007). Métricas de Similitud para Búsqueda Aproximada. *Revista de Tecnología - Journal of Technology • Volumen 6, No. 2*.

Garg, D., Trivedi, K., & B.B.Panchal. (2013). *A Comparative study of Clustering Algorithms using MapReduce in Hadoop.* International Journal of Engineering Research & Technology (IJERT).

Gartner, Inc. (2 de marzo de 2007). *'Dirty Data' is a Business Problem, Not an IT Problem, Says Gartner.* Obtenido de: http://www.gartner.com/newsroom/id/501733

Gartner, Inc. (2 de febrero de 2015). *Gartner Says CIOs and CDOs Must 'Digitally Remaster' Their Organizations.* Obtenido de: http://www.gartner.com/newsroom/id/2975018

GNU Aspell. (2014). Obtenido de http://aspell.net/

Gonzales-Cam, C. (2008). Algoritmos fonéticos en el desarrollo de un sistema de información de marcas y signos distintivos. *Biblios*, 32. Obtenido de http://www.redalyc.org/articulo.oa?id=16118981008

Hernández, M. (2013). *Clase Magistral. Curso Calidad de Datos.* San José: Universidad Cenfotec.

Hernández, R., Fernández, C., & Baptista, P. (2006). *Metodología de la investigación.* México: McGraw Hill.

IBM. (2014). Obtenido de IBM Knowledge Center: http://www-01.ibm.com/support/knowledgecenter/SSWSR9_11.0.0/com.ibm.mdmhs.dev.platform.doc/concepts/c_Phonetic_Key_Generators.html

Knuth, D. (1973). *The art of computer programming: sorting and searching.* Massachusetts: Addison-Wesley.

Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions and reversals. Sov. Phys. Dokl. 10, 8, 707–710. Original in Russian in Dokl. Akad. Nauk SSSR 163, 4, 845–848, 1965

Málaga, J. T., Vera, G., & Oliveros, R. (2008). Tipos, Métodos y Estrategias de Investigación Científica. *Revista de la Escuela de Posgrado*. Obtenido de http://www.imarpe.pe/imarpe/archivos/articulos/imarpe/oceonografia/adj_modela_pa-5-145-tam-2008-investig.pdf

McCallum, A., Nigam, K., & H., U. L. (2000). Efficient Clustering of HighDimensional Data Sets with Application to Reference Matching. *The sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 169-178.

Microsoft. (6 de octubre de 2014). *Comparación de SOUNDEX y DIFFERENCE*. Obtenido de http://technet.microsoft.com/es-es/library/ms189282(v=sql.105).aspx

Nielsen, P., Delaney, K., G, L., Machanic, A., Randal, P., & K, T. (2010). *SQL Server MVP Deep Dives.* Greenwich: Manning Publications Co.

OpenRefine. (16 de julio de 2014). *OpenRefine*. Obtenido de OpenRefine: https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth

Oracle. (2014). Obtenido de Oracle Database SQL Reference: http://docs.oracle.com/cd/B19306_01/server.102/b14200/functions148.htm

Pelakais, C., Finol, F. M., Noel, N., & José, B. O. (2005). *El ABC de la investigación: Una aproximación teórico-práctica.* Maracaibo.

Rico, L. (1999). Resumen del capítulo 4. En I. Montero, & O. León, *Diseño de Investigaciones.* Madrid: MacGraw Hill.

Storti, M., D'Elía, J., Paz, R., Dalcín, L., & Pucheta, M. (2012). *Algoritmos y Estructuras de Datos.*

Studer, R., Benjamins V., R., & Fensel, D. (1998). *Knowledge engineering: principles and methods.*Obtenido de http://www.it.iitb.ac.in/~palwencha/ES/Knowledge%20engineering%20-%20Principles%20and%20methods.pdf

Tayi, G., & Ballou, D. P. (1998). Examining Data Quality. *Magazine Communications of the ACM,* 54-57.

Wagner, R., Fischer, M. (1974). The String-to-String Correction Problem. *J. ACM* 21, 168-173.

12 Anexos

1. Código de programación de la función GDS.

```
public Coordenada3D ubicar(String cadena, boolean orderSensitive, double MovFraction) {
Coordenada3D ubicacion = new Coordenada3D(0d,0d,0d);
if(cadena != null && cadena.length() > 0){
        String normaCadena = StringUtils.removeAccents(cadena);
        normaCadena = StringUtils.removePunctCtrlChars(normaCadena);
                if(orderSensitive)  {
                String[] frags = org.apache.commons.lang.StringUtils.split(normaCadena);
            TreeSet<String> set = new TreeSet<String>();
            for (String ss : frags) {     set.add(ss);  }
            StringBuffer b = new StringBuffer();
            Iterator<String> i = set.iterator();
            while (i.hasNext()) {
               b.append(i.next());
               if (i.hasNext()) {    b.append(' ');     }
                  }
           normaCadena = b.toString();
        }
         normaCadena = StringUtils.asciify(normaCadena);
        char[] caracteres = normaCadena.toUpperCase().toCharArray();
        int num_chars = caracteres.length;
        boolean firstChar = true;
        for(int c = num_chars - 1; c >= 0; c-- ){
        Coordenada3D  nextcoor = qwertykeyboard.get(caracteres[c]);
                if(nextcoor != null) {
                        if( firstChar ){
                                ubicacion.setX(nextcoor.getX());
                                ubicacion.setY(nextcoor.getY());
                                ubicacion.setZ(nextcoor.getZ());
                                firstChar=false;
                        }
                        else {
                                ubicacion.fraction(nextcoor,MovFraction);
                        }
                 }
        }
 }
return  ubicacion;
}
```

2. Código de Indice NonClustered

```
USE [Corpus]
GO

--/ Object:  Index [idx_ Coordenadas]   Script Date: 15/02/2015 05:14:21 p.m. CREATE
NONCLUSTERED INDEX [idx_Coordenadas] ON [dbo].[CorpusLocated]
(
 [x] ASC
)
INCLUDE ([y],
 [z]) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
```

3. Plan de ejecución real del script #1 aplicado a 1 millón de registros.